5th Conference on Advances in Financial Technologies

AFT 2023, October 23-25, 2023, Princeton, NJ, USA

Edited by

Joseph Bonneau
S. Matthew Weinberg

LIPICS

Editors

Joseph Bonneau
New York University, NY, USA
jcb@cs.nyu.edu

S. Matthew Weinberg
Princeton University, NJ, USA
smweinberg@princeton.edu

ACM Classification 2012
Security and privacy → Mathematical foundations of cryptography; Theory of computation → Cryptographic primitives; Theory of computation → Cryptographic protocols; Security and privacy → Distributed systems security; Security and privacy → Privacy-preserving protocols; Security and privacy → Pseudonymity, anonymity and untraceability; Theory of computation → Algorithmic mechanism design; Applied computing → Economics; Applied computing → Digital cash

ISBN 978-3-95977-303-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-303-4.

Publication date
October, 2023

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.AFT.2023.0

LIPIcs – Leibniz International Proceedings in Informatics

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface

This volume contains 31 papers selected out of 100 submissions for the 5th Conference on Advances in Financial Technologies (AFT ‘23) held on October 23–25, 2023. Each paper received detailed reviews by several program committee members and external reviewers.

The conference was held at Princeton University in Princeton, NJ. Each accepted paper was presented via a 15-minute live presentation, followed by a 5-minute question/answer period with the audience.

Two full-day workshops were co-located with AFT on October 26. Aniket Kate and Andrew Miller co-organized the workshop on Decentralized Credit Networks (DCN), and Christian Cachin and Giuliano Losa co-organized the workshop on Heterogeneous Trust in Distributed Systems (HTDS).

We would like to thank all Program Committee members and subreviewers for their service in selecting the AFT program, and all authors for submitting their work for consideration. We are also grateful to the AFT steering committee, and especially Ittay Eyal, for their support and guidance throughout the process.

We are also extremely thankful to our industry sponsors, whose financial support is essential to running AFT:

- a16z Crypto (gold-level)
- Ava Labs (gold-level)
- IC3 (silver-level)
- IOHK (silver-level)
- DeCenter (silver-level)
- StarkWare (silver-level)

We also thank the IACR for granting AFT ‘in cooperation’ status.

Finally, we would like to thank all of the staff at Princeton University who made this event possible, and especially Elizabeth Wang, Michele Brown, and Mitra Kelly.

5th Conference on Advances in Financial Technologies (AFT 2023).
Editors: Joseph Bonneau and S. Matthew Weinberg
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Steering Committee

External Reviewers

List of Authors

Shresth Agrawal (14)
Technische Universität München, Germany

Orestis Alpos (31)
University of Bern, Switzerland

Guillermo Angeris (4)
Bain Capital Crypto, San Francisco, CA, USA

Alireza Arjmand (27)
University of Alberta, Edmonton, Canada

Sarah Azouvi (5)
Protocol Labs, San Francisco, CA, USA

Maryam Bahrani (21)
a16z Crypto, New York, NY, USA

Mahsa Bastankhah (3)
École Polytechnique Fédérale de Lausanne, Switzerland

Carsten Baum (12)
Technical University of Denmark, Lyngby, Denmark

Donald Beaver (7)
Independent Scholar, Pittsburgh, PA, USA

George Bissias (6)
University of Massachusetts Amherst, MA, USA

Dan Boneh (18, 26, 29)
Stanford University, CA, USA

Andrea Canidio (24)
CoW Protocol, Lisbon, Portugal

Konstantinos Chalkias (7)
Mysten Labs, Palo Alto, CA, USA

James Hsin-yu Chiang (10, 11, 12)
Aarhus University, Denmark

Tarun Chitra (4)
Gauntlet, New York, NY, USA

Nicolas Christin (8)
Carnegie Mellon University, Pittsburgh, PA, USA

Marko Cirkovic (16)
University of Bern, Switzerland

Jeremy Clark (22)
Concordia University, Montreal, Canada

Bernardo David (10, 11, 12)
IT University of Copenhagen, Denmark

Ladi de Naurois (7)
Washington DC, USA

Theo Diamandis (4)
MIT CSAIL, Cambridge, MA, USA

Stefan Dziembowski (17)
University of Warsaw, Poland; IDEAS NCBR, Warsaw, Poland

Vero Estrada-Galiñanes (3)
École Polytechnique Fédérale de Lausanne, Switzerland

Alex Evans (4)
Bain Capital Crypto, San Francisco, CA, USA

Ittay Eyal (10)
Technion, Haifa, Israel

Zhou Fan (25)
Harvard University, Cambridge, MA, USA

Edward W. Felten (23)
Offchain Labs, Washington, D.C., USA

Bryan Ford (3)
École Polytechnique Fédérale de Lausanne, Switzerland

Elijah Fox (19)
Duality Labs, New York, NY, USA, USA

Tore Kasper Frederiksen (12)
Zama, Paris, France

Robin Fritsch (24)
Cow Protocol, Lisbon, Portugal; ETH Zürich, Switzerland

Mariana Gama (11)
COSIC, KU Leuven, Belgium

Pranav Garimidi (21)
a16z Crypto, New York, NY, USA

Ben Glickenhaus (6)
University of Massachusetts Amherst, MA, USA

Daniel Goldman (22)
OffchainLabs, Princeton, NJ, USA

Tiantian Gong (10)
Purdue University, West Lafayette, IN, USA

Gregory Griffith (6)
Bitcoin Unlimited

Tivas Gupta (20)
Special Mechanisms Group, USA

Lioba Heimbach (9)
ETH Zürich, Switzerland

Simon Holmgaard Kamp (31)
Aarhus University, Denmark

Daisuke Kawai (8)
Carnegie Mellon University, Pittsburgh, PA, USA

Mahimna Kelkar (7, 23)
Cornell University, New York City, NY, USA

Patrik Keller (6)
Universität Innsbruck, Austria

Majid Khabbazian (15, 27)
University of Alberta, Edmonton, Canada

William Knottenbelt (16)
Imperial College London, UK

Lefteris Kokoris-Kogias (7)
Mysten Labs, London, UK; IST Austria, Klosterneuburg, Austria

Paweł Kędzior (17)
University of Warsaw, Poland

Duc V. Le (16)
Visa Research, Sunnyvale, CA, USA

Christian Janos Lebeda (11)
IT University of Copenhagen, Denmark; Basic Algorithms Research Copenhagen, Denmark

Kevin Lewi (7)
Meta Platforms, Inc., Menlo Park, CA, USA

Akaki Mamageishvili (23)
Offchain Labs, Zürich, Switzerland

Yacov Manevich (1)
IBM Research - Zürich, Switzerland

Francisco Marmolejo-Cossio (25)
Harvard University, Cambridge, MA, USA; IOG, USA

Louis-Henri Merino (3)
École Polytechnique Fédérale de Lausanne, Switzerland

Barnabé Monnot (30)
Ethereum Foundation, Berlin, Germany

Mahsa Moosavi (22)
Concordia University, Montreal, Canada; OffchainLabs, Princeton, NJ, USA

Daniel Moroz (25)
Harvard University, Cambridge, MA, USA

Danielle Movsowitz Davidow (1)
Tel-Aviv University, Israel

Joachim Neu (14)
Stanford University, CA, USA

Michael Neuder (25)
Harvard University, Cambridge, MA, USA

Stephen H. Newman (13)
Princeton University, NJ, USA

Wilson D. Nguyen (18)
Stanford University, CA, USA

Jesper Buus Nielsen (31)
Aarhus University, Denmark

Valeria Nikolaenko (7)
a16z crypto, Palo Alto, CA, USA

Mallesh M. Pai (19, 20)
Department of Economics, Rice University, Houston, TX, USA; Special Mechanisms Group, USA

Jennifer Pan (30)
Jump Crypto, Chicago, IL, USA

David C. Parkes (25)
Harvard University, Cambridge, MA, USA

Aditi Partap (26)
Stanford University, CA, USA

Ziyan Qu (3)
École Polytechnique Fédérale de Lausanne, Switzerland

Rithvik Rao (25)
Harvard University, Cambridge, MA, USA

Max Resnick (19, 20)
Special Mechanisms Group, USA

Lior Rotem (26)
Stanford University, CA, USA

Tim Roughgarden (21)
a16z Crypto , New York, NY, USA; Columbia University, New York, NY, USA

Bryan Routledge (8)
Carnegie Mellon University, Pittsburgh, PA, USA

Arnab Roy (7)
Mysten Labs, Palo Alto, CA, USA

Fahad Saleh (30)
Wake Forest University, Winston Salem, NC, USA

Mehdi Salehi (22)
OffchainLabs, Princeton, NJ, USA

Eric Schertenleib (9)
ETH Zürich, Switzerland

Jan Christoph Schlegel (23)
City, University of London, UK

Caspar Schwarz-Schilling (30)
Ethereum Foundation, Berlin, Germany

Srinath Setty (18)
Microsoft Research, Redmond, WA, USA

Nihar Shah (30)
Jump Crypto, Chicago, IL, USA

Alberto Sonnino (7)
Mysten Labs, London, UK; University College London, UK

Kyle Soska (8)
Ramiel Capital, New York, NY, USA

Ertem Nusret Tas (14, 29)
Stanford University, CA, USA

Thomas Thiery (30)
Ethereum Foundation, Lyon, France

Eran Toch (1)
Tel-Aviv University, Israel

Mohammad Amin Vafadar (15)
University of Alberta, Edmonton, Canada

Saravanan Vijayakumaran (28)
Department of Electrical Engineering, Indian Institute of Technology Bombay, Mumbai, India

Marko Vukolić (5)
Protocol Labs, San Francisco, CA, USA

Xuechao Wang (5)
Thrust of Financial Technology, HKUST(GZ), Guangzhou, China

Zhipeng Wang (16)
Imperial College London, UK

Roger Wattenhofer (9)
ETH Zürich, Switzerland

Aviv Yaish (2)
The Hebrew University of Jerusalem, Israel

Ariel Zetlin-Jones (8)
Carnegie Mellon University, Pittsburgh, PA, USA

Haoqian Zhang (3)
École Polytechnique Fédérale de Lausanne, Switzerland

Dionysis Zindros (14)
Stanford University, CA, USA

Aviv Zohar (2)
The Hebrew University of Jerusalem, Israel

Privacy-Preserving Transactions with Verifiable Local Differential Privacy

Danielle Movsowitz Davidow[1]
Tel-Aviv University, Israel

Yacov Manevich
IBM Research – Zürich, Switzerland

Eran Toch
Tel-Aviv University, Israel

Abstract

Privacy-preserving transaction systems on blockchain networks like Monero or Zcash provide complete transaction anonymity through cryptographic commitments or encryption. While this secures privacy, it inhibits the collection of statistical data, which current financial markets heavily rely on for economic and sociological research conducted by central banks, statistics bureaus, and research companies. Differential privacy techniques have been proposed to preserve individuals' privacy while still making aggregate analysis possible. We show that differential privacy and privacy-preserving transactions can coexist. We propose a modular scheme incorporating verifiable local differential privacy techniques into a privacy-preserving transaction system. We devise a novel technique that, on the one hand, ensures unbiased randomness and integrity when computing the differential privacy noise by the user and on the other hand, does not degrade the user's privacy guarantees.

2012 ACM Subject Classification Security and privacy → Privacy-preserving protocols; Security and privacy → Trust frameworks

Keywords and phrases Differential Privacy, Blockchain, Privacy Preserving, Verifiable Privacy

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.1

Related Version *Full Version*: `https://eprint.iacr.org/2023/126`

Supplementary Material *Software (Source Code)*: `https://github.com/yacovm/ZKAT-VDP`
archived at `swh:1:dir:48554d7cfe172767c418e2e8f506fde1200c3cdd`

Funding This work was supported by a grant from the Tel Aviv University Center for AI and Data Science (TAD).

Acknowledgements We would also like to thank Dany Moshkovich and the reviewers of this paper for their helpful comments and thorough review.

1 Introduction

1.1 Motivation

The issue of privacy holds significant importance and poses ongoing challenges in blockchain systems. This concern has become a substantial barrier for traditional financial institutions seeking to adopt blockchain technologies [26]. Initially, blockchain protocols offered only limited pseudo-anonymity. However, user concerns have given rise to privacy-preserving blockchain systems such as Monero [42], ZCash [4], and Twilight [20]. These systems aim to increase users' anonymity and conceal transaction details. For instance, ZCash [4]

[1] The work was done during an internship at IBM Research – Haifa.

enables users to encrypt transaction data, thereby safeguarding the privacy of the sender, the recipient, and the transaction amount. Additionally, ZCash ensures the validity of transactions without revealing any supplementary information beyond the transaction's legitimacy. These advancements within blockchain systems show a proactive approach to addressing privacy concerns, facilitating the broader adoption of blockchain technology.

However, many pivotal applications such as statistics gathering [29], federated model learning [32, 31], and anomaly analysis [30] necessitate the acquisition of aggregated models from the shared data of numerous users while still maintaining user confidentiality. While such aggregate and statistical models can yield benefits to the market as a whole [43], it is equally crucial to ensure that data collection is conducted to minimize privacy threats to the users. In particular, we aim to inhibit the ability of the data analyst to link multiple data points of the user [19] or retrace their real identity via third-party datasets [16]. If these risks aren't adequately addressed, users may opt to evade participation in data collection initiatives or provide misleading information that could lead to biased models.

Central Bank Digital Currencies (CBDCs) present a significant use case for the aggregation of statistical models. Central banks across the globe have exhibited a growing interest in developing and issuing CBDCs, which represent a digitized form of a nation's fiat currency and are intended to be widely available to the general public. Blockchain-based CBDCs are designed to align with the requirements of regulated financial institutions. Consequently, these institutions favor *permissioned* blockchain platforms that mandate identity management, audibility, and non-deniability to comply with government-established monetary regulations.

However, the transactions resulting from existing privacy-preserving systems tailored for financial institutions [3] appear random, preventing the extraction of meaningful insights through observation. They prioritize the properties of *unlinkability* and *untraceability* to protect user privacy. Unlinkability ensures that user actions cannot be easily linked together, while untraceability maintains the anonymity of each transaction's sender and recipient. These systems safeguard sensitive user information and transaction history by upholding these properties. Therefore, central banks cannot derive valuable insights and may struggle to meet regulatory requirements.

Local differential privacy [33] potentially solves this by adding random "noise" to the data. This noise ensures that no specific user can be identified from the processed data, a privacy guarantee laid out by Dwork et al. (2006) [23]. Thus, financial institutions can gather and analyze aggregated data without risking the exposure of sensitive information tied to individual transactions. However, a challenge arises here: while local differential privacy empowers users to introduce noise to their data, some may distort this feature by injecting biased noise into it, which may adversely affect the integrity of the information collected [11, 14]. Therefore, in blockchain environments, it is crucial to have a mechanism verifying the correctness of introduced noise when implementing local differential privacy.

1.2 Verifiable Differentially Private Transactions

In this paper, we introduce the Verifiable Differentially Private (VDP) transaction scheme, designed to ensure user privacy and data integrity during the data collection process for aggregated models. Our innovative scheme bolsters the capabilities of any privacy-preserving transaction system that maintains unlinkability and untraceability, such as Zcash [4]. With this enhanced setup, a third-party entity, like a data analyst, can gather aggregated data while simultaneously preserving user privacy and supplying plausible deniability.

Our approach stands out from other privacy-preserving transaction methodologies by incorporating *verifiable* Local Differential Privacy (LDP) utilizing zero-knowledge proofs. This integration forms an integral part of the privacy-preserving transaction system. By adopting LDP methods [33], data analysts, such as the statistical bureau of a central bank collecting financial information, can provide users with plausible deniability for their data that can be represented as binary variables. These variables can encompass a wide range of characteristics, such as non-profit organization affiliation or location in the U.S. Furthermore, including zero-knowledge proofs allows data analysts to verify the accuracy and integrity of the disclosed data.

Local Differential Privacy (LDP) approaches typically require users to generate random numbers that are then used to create noise in the data [33]. Keeping these random numbers confidential is crucial for maintaining user privacy. At the same time, it's also important that the process of creating random numbers and generating noise from them is unbiased. This means that even if someone involved in the process – either the user or the data analyst – has malicious intent, the LDP approach should still meet the following requirements: (i) The generation of noise cannot be influenced by either the user or the data analyst; (ii) The user can provide proof, without compromising privacy, that the noise is not biased; (iii) Once the random numbers for a particular input are calculated, the user cannot change these numbers or add a different level of noise. To meet these requirements, we introduce the VerRR algorithm, which is a straightforward verifiable LDP mechanism based on the concept of 'randomized response' [24].

1.3 Related Work

In the following sections, we outline the different approaches for verifiable differential privacy employed by various works and highlight the differences from our technique.

1.3.1 Central Differential Privacy

When applying Differential Privacy (DP) techniques for preserving privacy, traditional methods often rely on the assumption that the process of noise generation and application is intrinsically reliable. However, recent scholarly work [12, 13] has underscored the naivety of this assumption. These studies expose DP's vulnerability to a range of manipulation attacks, thereby threatening the integrity of the data under examination.

Several studies have proposed strategies to counter various manipulation attacks. The key aim across all these works is to generate and apply the noise in a way that can be verified. Within the centralized DP model, studies such as [6, 38, 41] have zeroed in on a series of use cases. In these instances, the data owners forward their data – either in plain text, encrypted or in a secret-shared format – to a single party or a consortium of parties who together form an entity termed a "curator". This curator subsequently applies noise and generates an anonymized, privacy-preserving data set suitable for further analysis.

Secure Multi-Party Computation (sMPC). In the context of secure multi-party computation (sMPC), a user secretly shares their input and distributes the shares across several servers. As proposed in [6], if at least one of the servers collecting the data is honest, the resulting computation is either correct or detected to be incorrect. Unfortunately, such a single-client-multi-server model does not fit the use case of our work since we operate under the assumption that the data analyst, who also serves as the curator, could potentially have malicious intentions and is not divided into separate entities.

Zero-Knowledge Proofs. In the study by Narayan et al., the curator creates a database comprising different users' data elements and subsequently publishes a cryptographic commitment to the entire database [38]. When a data analyst sends a query interested in specific statistical properties, the curator executes the differential privacy function. In addition, the curator generates[2] a Non-Interactive Zero-Knowledge (NIZK) proof, validating that the result aligns with the prior commitment, and sends both the result and the proof back to the data analyst. The data analyst then verifies the NIZK against the commitment to ensure that the differential privacy mechanism was correctly executed.

The most notable downside of the naive zero-knowledge approach employed by previous works is that it is assumed that the noise was sampled correctly and without bias. However, if the party that produces the NIZK is malicious, it may use far from random noise and thus either harm the privacy of the users or skew the results towards a value it favors. In contrast, in our technique, we ensure that some of the randomness is sampled by the user itself (unlike the work of [41] where the curator randomly samples the randomness) and some by the analyst, thus producing noise that is non-biased.

1.3.2 Local Differential Privacy Mechanisms

LDP mechanisms allow for developing a randomized response that is operated locally, making it fit for distributed blockchain environments. However, ensuring data integrity is a more challenging task since data manipulation occurs locally. Research, such as [2, 35], concentrates on making adjustments to existing LDP mechanisms to make them verifiable. This allows data analysts to verify how users introduce noise to the data.

The study by Kato et al. employs cryptographic randomized response techniques to validate existing LDP mechanisms. Their approach ensures the complete execution of privacy protection on the client side, without sacrificing user privacy [35]. However, their method's primary drawback is its interactive nature. Interactive methods are typically more time-consuming and resource-intensive than their non-interactive counterparts. They necessitate several communication exchanges and direct engagement with the data analyst. Furthermore, interactive methods do not support throttling. Hence, during periods of high network usage, while non-interactive systems can manage the demand by forming a queue that will eventually be cleared unbeknownst to the users, interactive systems may freeze, forcing users to endure an uncertain waiting time. Our proposed scheme, on the other hand, is non-interactive, overcoming these limitations.

Garrido et al. follow an approach based on zero-knowledge Succinct Non-interactive ARgument of Knowledge (zk-SNARKs) to adapt the implementation of select LDP mechanisms to a verifiable computation technique to prove the correctness of a differentially private query output [36]. However, their technique does not uphold the unlinkability and untraceability properties needed in a privacy-preserving transaction system. These properties are not upheld since the prover (i.e., the user) signs the randomness used as the base of the LDP mechanism with their private key, and the verifier (i.e., the data analyst) needs to know the prover's public key to verify the integrity of the response. By knowing the prover's public key, the data analyst can later connect the generated randomness and the transfer it was used in, thus revealing the identity of the user making the transfer.

Outline. The remainder of the paper is structured as follows. In Section 2, we introduce some preliminary concepts and provide the relevant background regarding blockchain-based privacy-preserving transaction systems and verifiable differential privacy. We present a

[2] For ease of explanation, we mention the curator as the producer of the NIZK, but in the cited study, this task is delegated to a separate party: the analyst.

high-level overview of our scheme in Section 3, and dive into the scheme's full details in Section 4. In Section 5, we evaluate the performance of our scheme. We discuss design choices in Section 6 and summarize our contribution in Section 7.

2 Preliminaries

2.1 Blockchain-Based Privacy-Preserving Transactions

Blockchain transaction systems are typically modeled using two approaches: (i) The account model, used in Ethereum [46]. This model is more user-friendly as it presents an abstraction of accounts and balances; (ii) The Unspent Transaction Output (UTXO) model, used in Bitcoin [37] and Zcash [4]. This model presents an abstraction of a plurality of coins, each with a balance and a condition for spending[3]. While it is considered an open problem how to "hide" a transaction's sender in an account model [28] [4], "hiding" the sender in the UTXO model is easy [4]. Thus, this work focuses on the UTXO model.

In a privacy-preserving transaction system using the UTXO model, when a sender transfers funds to a recipient, to preserve the privacy of the transaction, three things need to be hidden: (i) The sender's identity, (ii) The recipient's identity, and (iii) The transferred amount. The transferred amount is usually hidden through a cryptographic commitment and a zero-knowledge proof proving that the sum of input coins is greater or equal to the sum of the output coins. The recipient's anonymity is ensured by creating outputs that either have one-time public keys [42] or by hiding the output token's identity by having the coin itself be a commitment [4] to its properties (e.g., owner, amount). Maintaining the sender's anonymity varies widely across techniques; Monero [42] uses ring signatures to hide the source address among a randomly picked set of potential source addresses, while Zcash uses a Merkle tree and zero-knowledge proof of membership to obscure the spent token.

Our work is based on Zcash's approach to concealing sender, recipient, and amount but is flexible enough to incorporate other privacy-preserving transaction systems that encode the entire token as a commitment. This includes the model proposed by [3], which is better suited for CBDCs and permissioned settings, replacing the Merkle tree membership proof with proof of knowledge of a signature of an entity that certifies the coin. Zcash terminology is used throughout the paper.

2.2 Differential Privacy

Differential Privacy (DP) [23, 22] is a formal notion of privacy designed to allow learning helpful information about a population while learning as little as possible about an individual. DP guarantees that the presence or absence of any specific individual in a dataset does not affect the query output results of that database. To achieve this privacy requirement, DP models change the data by using some randomness that masks the user identity.

There are two main models of DP: the centralized model and the local model. In the centralized model, sensitive data is collected centrally in a single dataset, and a trusted data curator can access the raw data. Each user sends their data to the curator without noise

[3] In most cases, presenting a signature that is verifiable under a public key that its hash is encoded in the coin.

[4] In an account model, hiding the sender involves hiding it within a K sized anonymity set [10], while a UTXO based approach hides the sender within all possible senders in the system.

in this model. Since we assume analysts requesting access to this dataset are malicious, the curator is responsible for correctly executing the differentially private mechanisms the analysts specify.

▶ **Definition 1** ($\epsilon-$differentially privacy [22]). *A randomized algorithm $\mathcal{K}$ is $\epsilon-$differentially private if for all data sets D_1 and D_2 differing on at most one element, and all $S \subseteq Range(\mathcal{K})$, $P[\mathcal{K}(D_1) \in S] \leq e^{\epsilon} P[\mathcal{K}(D_2) \in R]$.*
The probability is taken over the coin tosses of $\mathcal{K}$.

Local DP models can be based on a *randomized response* mechanism as was first proposed by Warner in 1965 [45] and formalized in the context of learning by Kasiviswanathan et al. [34]. In this model, the data curator and the analyst are often the same, and no trusted third party exists that holds the data and executes the mechanism. Therefore, the user makes data differentially private before sending it to the analyst, ensuring that even if an adversary gains access to the personal responses of individuals in the database, it will not be able to learn much about the individual's data.

▶ **Definition 2** (Local randomizer [24]). *An $\epsilon-$local randomized $R : \mathcal{X} \to W$ is an $\epsilon-$differentially private algorithm that takes a database of size $n = 1$ as input.*

We incorporate local DP techniques into our scheme since our work is based on a privacy-preserving transaction system in which we assume that any parties (i.e., the users or the data analyst) may be dishonest but not both. Therefore, an honest user cannot trust the analyst to properly blind their data without leaking it.

2.3 Randomized Response

Dwork and Roth presented in [24] a variant of the randomized response mechanism, in which a user answers a "Yes" or "No" question as follows:

1. Flip a coin.
2. If **tails**, then respond truthfully.
3. If **heads**, then flip a second coin and respond "Yes" if heads and "No" if tails.

In this algorithm, the user flips two coins before returning the response. Flipping two coins creates uncertainty about the true answer. This uncertainty is the source of privacy, as it gives plausible deniability to the data subject. In section A in the appendix of our full version [18], we prove the randomized response algorithm described above satisfies $\ln 3$-differentially privacy.

We note that when using the randomized response technique, the number of "Yesses" and "Noes" depends on the result of tossing the coin once or twice. Therefore, if an analyst wants to estimate the true number of "Yesses", it would need to analyze the randomness in the randomized response algorithm and estimate how many of the "Yesses" are truthful responses and how many are "fake", and are a result of the random coin flips.

2.4 Basic Cryptographic Building Blocks

The following section extensively explains zero-knowledge proofs, which are crucial in our scheme. We denote the symbol λ to represent the system-wide security parameter.

2.4.1 Commitment Schemes

A commitment scheme is a protocol between a sender and a receiver defined by three probabilistic polynomial time algorithms: $\langle Setup, Commit, Open\rangle$. The sender and receiver each invoke the *Setup* operation with the desired security parameter and then get back public parameters to be used for *Commit* and *Open*. The sender uses the *Commit* operation to encode a message m and get back a *commitment* to m. The sender then sends the commitment to the receiver. At a later stage, the sender may convince the receiver that the commitment encodes the message m by interacting with the receiver via the *Open* operation. In order for the commitment scheme to be useful, it needs to satisfy two security properties:

- **Hiding**: For any probabilistic polynomial time adversary $\mathcal{A}$ there exists a negligible[5] function ϵ such that for every two messages m_0, m_1 chosen by $\mathcal{A}$ and a commitment $cm \leftarrow Commit\left(m_b\right)$ for $b \in \{0,1\}$ it holds that: $Pr\left[b \leftarrow \mathcal{A}\left(pp, cm\right)\right] - \frac{1}{2} \leq \epsilon\left(\lambda\right)$.
- **Binding**: For any probabilistic polynomial adversary $\mathcal{A}$ there exists a negligible function ϵ such that for every m and $cmt \leftarrow Commit\left(m\right)$ and every $m' \neq m$ chosen by $\mathcal{A}$: $Pr\left[Open\left(pp, cmt, m\right) = m'\right] \leq \epsilon\left(\lambda\right)$.

A commitment can either be information-theoretic binding or hiding, and it can be computationally binding or hiding (but cannot be both information-theoretic binding and hiding). In our scheme, the commitment does not need to be homomorphically additive or have any special property, and as such, it is completely pluggable.

2.4.2 Digital Signatures

A digital signature scheme is a triple $\langle Gen, Sign, Verify\rangle$ of probabilistic polynomial time algorithms. $Gen(1^\lambda)$ outputs (sk, pk) a private key and a public key respectively. For a message m, $Sign\left(m, sk\right)$ outputs a signature σ. For a message m, a public key pk and a signature σ, $Verify\left(pk, m, \sigma\right)$ accepts or rejects. The security property we require from a signature scheme is:

- **Unforgability**: For every probabilistic polynomial time adversary $\mathcal{A}$ with access to a signing oracle $\mathcal{O}$ which replies to queries denoted by a set Q there exists a negligible function ϵ such that $Pr\left[Verify\left(pk, m, \sigma\right) = 1 \wedge m \notin Q\right] \leq \epsilon\left(\lambda\right)$.

As in the case of the commitment scheme, the signature scheme employed by our protocol can vary, and its choice is insignificant.

2.4.3 Public Key Encryption

An encryption scheme is a triple $\langle Gen, Enc, Dec\rangle$ of probabilistic polynomial time algorithms. $Gen(1^\lambda)$ outputs (sk, pk) a private key and a public key respectively. For a message m, $Enc\left(m, pk\right)$ outputs a ciphertext c. Given a ciphertext c and a private key sk, $Dec\left(c, sk\right)$ outputs a message m. For every public key pk with a corresponding sk and message m it holds that $Pr\left[Dec\left(Enc\left(m, pk\right), sk\right) = m\right] = 1$. The security property we require from an encryption scheme is:

- **Indistinguishablility**: For every probabilistic polynomial time adversary $\mathcal{A}$ and every pk generated by $Gen(1^\lambda)$ and for every two messages m, m' there exists a negligible function ϵ such that $Pr\left[\mathcal{A}\left(Enc\left(m, pk\right)\right) = 1\right] - Pr\left[\mathcal{A}\left(Enc\left(m', pk\right)\right) = 1\right] \leq \epsilon\left(\lambda\right)$.

[5] A function $\epsilon : \mathbb{N} \rightarrow \mathbb{R}$ is negligible if for every positive polynomial $p\left(n\right)$ there exists an N such that $\forall n > N$: $\epsilon\left(n\right) < \frac{1}{p(n)}$.

Our protocol does not require any stronger property for the encryption scheme, such as resistance against chosen ciphertext attacks (CCA1 and CCA2).

2.5 Zero-Knowledge Proofs

For a language $\mathcal{L} \in NP$, denote the relation $R_{\mathcal{L}}$ to be the pairs (x, w) of statements and witnesses for x being in $\mathcal{L}$.

A zero-knowledge proof is a protocol between a prover P and a verifier V in which the prover can convince the verifier that it knows a witness w for a statement x if and only if $(x, w) \in R_{\mathcal{L}}$, and the verifier learns nothing from the protocol.

More formally, a pair of algorithms (P, V) is a zero-knowledge proof system for a language $\mathcal{L}$ if the following three conditions hold:

- **Completeness**: For every $x \in \mathcal{L}$ there exists w such that $Pr\left[\langle P(w), V\rangle(x) = 1\right] = 1$.
- **Soundness**: For every $x \notin \mathcal{L}$ and every P^* and every w there exists a negligible function ϵ such that $Pr\left[\langle P^*(w), V\rangle(x) = 1\right] \leq \epsilon\left(\lambda\right)$.
- **Zero-knowledge**: For every probabilistic polynomial time V^* there exists a probabilistic polynomial time simulator S such that for every $x \in \mathcal{L}$: $View^{P}_{V^*}\left(x\right) \equiv S\left(x\right)$.

While zero-knowledge proofs are not restricted to statements belonging to languages in NP, they are often used for such, especially for languages where verification of $(x, w) \in R_{\mathcal{L}}$ is efficient and can be modeled as a boolean or arithmetic circuit, but an efficient algorithm to find w for a random x is not known. A prominent example is: $(x, w) \in R_{\mathcal{L}} \Leftrightarrow f\left(w\right) = x$ where f is a cryptographic hash function.

In an anonymous set membership, given a set of items found as leaves in a Merkle tree, one can prove knowledge of an item in the Merkle tree by proving in zero-knowledge a path comprised of hash pre-images from one of the leaves to the root of the Merkle tree. Such a technique is the cornerstone behind the privacy-preserving cryptocurrency Zcash [4]. In Zcash, a sender wishing to hide the payment source simply creates a zero-knowledge proof that they use a coin that was added to a Merkle tree. The Merkle tree's leaves are all coins added as part of past transactions.

Zero-knowledge proof schemes come in different forms, each with its strengths and weaknesses. In this work, we focus on schemes ideal for privacy-preserving payments, and thus we require the scheme to be non-interactive, efficiently verifiable, and have a small proof size (succinct). A scheme with such properties is termed a zk-SNARK (Zero-Knowledge Succinct Non-interactive ARgument of Knowledge).

3 Overview of the VDP Transaction Scheme

Our privacy-preserving, verifiable, and differentially-private transaction scheme expands the functionality of any given privacy-preserving transaction system (e.g., Zcash [4]) by enabling a third party (e.g., a data analyst) to collect information about transaction attributes while preserving user privacy.

Before elaborating on the participants, components, and transaction flow, we reiterate the scheme's primary objective. The two main actors in our setting are a **user** with potentially sensitive information they are unwilling to disclose without plausible deniability and a **data analyst** who aims to create a statistical model by aggregating information gathered from user transactions. The users conduct transactions of a privacy-preserving nature. These transactions reveal nothing about the transactor's identity, the recipient's identity, or the properties of the transaction (such as the amount transferred). Each transaction a user performs is accompanied by an additional Differentially Private Attribute (DPA) which results from applying an LDP algorithm to the transaction's private data[6]. The DPA can embed details like the sender's non-profit status, enabling the data analyst to tally transactions made by Blockchain users from both profit and non-profit sectors.

An LDP mechanism generates the differentially-private attribute that involves sampling randomness and using it for noise generation. The randomness must be kept secret from anyone but the user itself, lest a curious analyst can peel off the noise and be able to reconstruct the original value. At the same time, the scheme should force the user to generate randomness in a non-biased manner. Suppose the user is free to choose the randomness it uses. In that case, it can manipulate the result of the LDP mechanism making the data too "noisy", thus affecting the integrity of the data collected by the analyst.

Consequently, we seek a scheme with the following properties: (i) Neither the user nor the analyst can bias the generation of the noise used for the LDP mechanism; (ii) The user can prove in a privacy-preserving manner that the noise is non-biased; (iii) Once the randomness used to create the transaction's noise is computed, the user cannot compute new randomness or add a different amount of noise. We prove that our scheme fulfills all three aforementioned properties in Section 4.4.1. Another essential property of our scheme is preserving user privacy, meaning that once the transaction is complete, the analyst cannot identify the user who initiated the transaction. We prove this property in Section 4.4.2.

3.1 Participants

Our scheme involves the following participants:

Users. Users can exchange tokens within the system using transfer transactions, which are recorded on the blockchain. We presume that users possess specific individual attributes that they wish to disclose confidentially to the data analyst. These characteristics can cover a wide range of areas, for instance, signifying if the user is based in the U.S., is exempt from tax, or any other pertinent information that can be denoted as a binary variable.

Data analyst. This entity has the authority to collect aggregated information from the private attributes and activities of the users in the system. We assume that the analyst is only interested in analyzing data using statistical models regarding the system as a whole and is not interested in learning about specific individuals in the system.

Registration authority. This is a privileged entity in charge of registering all system users. For each user, it issues a long-term credential (i.e., a certificate) that binds the user's public key to its attributes. The same attributes will later be used as input to the LDP algorithm.

3.2 Threat Model

Our scheme is designed as a protocol that considers the interaction between two parties, the user and the data analyst, both of whom may exhibit dishonest behavior and possess conflicting objectives.

[6] In this work, we focus on the randomized response mechanism applied to one binary attribute. However, we argue that our ideas can be extended to randomized response vectors applied to multiple and non-binary attributes.

The first threat arises from the data analyst's desire to compute aggregated information derived from the user's transactions while potentially disregarding the user's privacy. Indeed, the analyst may deviate from the protocol in an attempt to link between the transaction and the user who submitted it [5].

The second threat arises from users who prioritize their privacy at the expense of providing misleading or corrupted data, which can undermine the integrity of the analysis conducted by the analyst. There are two primary ways in which users can launch such attacks: (i) Users can intentionally introduce randomness or noise manipulatively, rendering the data unreliable for generating unbiased and trustworthy aggregate statistical estimations [15]; (ii) Data poisoning attacks targeted at DP mechanisms. Data poisoning attacks involve adversarial manipulation of the input in order to influence the final aggregate result.

Since the analyst is interested in collecting statistics over attributes that correspond to the users themselves and for each transaction, such as whether the user is a non-profit or not or whether the user is based in the U.S., it is crucial that a user cannot mutate their attributes as they see fit. We guarantee this by requiring identities to be issued by the registration authority. Therefore, our scheme operates in the permissioned setting, whereas in a permissionless setting, each user is free to choose their own attributes.

Both the user and the analyst may deviate from the protocol. This introduces an element of uncertainty, as a party adhering to the protocol cannot definitively determine if the other party is also following protocol or diverging from it. Therefore, we aim to devise a protocol that protects the interest of any of the two parties as long as that party is honest. We note that such a protocol would fit the aforementioned threat model where both parties can deviate from the protocol, as each party can be assured that if it correctly follows the protocol, it will be protected from the misbehavior of the counter-party.

As we will see in the security analysis in Section 4.4, the focus on protecting the interests of honest parties lets us assume knowledge about the probability distribution of certain messages sent by honest parties, which then will set the probability distribution of messages sent by the counter-party regardless of whether it incorrectly samples its randomness.

In our protocol, the user initiates the protocol by sending a message to the data analyst, and the data analyst responds with its message. We consider a scenario where the data analyst does not respond to the user's message or sends back information that the user considers invalid as a scenario in which the data analyst aborts the protocol. Similarly to [3, 4], we consider network-level privacy out of scope and assume that the analyst cannot de-anonymize the user from inspecting the source of its network connection.

3.3 Components

Our scheme comprises three modular components: (i) A protocol to obtain and bind randomness. (ii) A verifiable LDP mechanism. (iii) An expanded privacy-preserving transfer that includes verifiable differentially-private data.

Obtain and bind randomness protocol. This protocol is a privacy-preserving verifiable joint random number generation protocol between the user and the analyst. The protocol uses a serial number created by the user to bind a set of unspent input random seeds used in a specific transfer to their corresponding jointly generated randomness. The analyst uses this serial number to verify that only one randomness is created for each set of unspent inputs.

Verifiable LDP mechanism. This mechanism makes the user's attributes differentially private before they are disclosed to the analyst. For simplicity, in this work, we use the basic *randomized response* mechanism described in Section 2.3 to make a single binary attribute differentially private.

Expanded privacy-preserving transfer. Our verifiable differentially private (VDP) transfer expands the transfer transaction defined by the underlying privacy-preserving transaction system. As we explained in Section 2.1, our scheme works with any transfer mechanism that: (i) Encodes tokens as commitments to properties of the token (token value, owner, and random seed are all part of the input to a cryptographic commitment scheme). (ii) Uses serial number exposure as a double-spending prevention (for a random seed ρ, the serial number is $PRF(\rho)$).

To verify the correctness of the randomness used as the source of randomness in the LDP mechanism and to ensure that the analyst cannot link a specific transfer to its sender, our VDP transfer uses two serial numbers. The user creates the first serial number during the *obtain and bind randomness* protocol. The second serial number, also created by the user, is used to prove the correctness of the randomness used by the verifiable LDP mechanism. From the analyst's point of view, the second serial number will only be accepted if the first serial number was previously observed (without being able to link the two together).

The analyst can verify that it has previously seen the first serial number because both numbers have the same precursor, a set of unspent input random seeds. The generated serial numbers satisfy the following security properties: (i) They are collision resistant – two different sets of unspent tokens produce two different serial numbers. (ii) They are deterministic – the same set of unspent tokens will always produce the same serial number. (iii) They are unforgeable – only the user who owns the unspent tokens can produce a valid serial number. Although both serial numbers are computed on the same set of unspent inputs, the analyst cannot link them to each other thanks to their construction. The unspent input seeds are passed through Pseudo-Random functions with different keys and hence are computationally unlinkable.

Additionally, our transfer uses zk-SNARK proofs to verify the integrity of the data disclosed by the user (i.e., the DPA). The proofs prove that the disclosed DPA matches the user's original attributes and is created using the jointly generated randomness as the base of the randomness used in the LDP mechanism.

3.4 VDP Transaction Flow

The VDP transaction scheme is illustrated in Figure 1. At first, the user contacts the registration authority (1.1) and uses a registration protocol to get a long-term credential (1.2). From then on, the user can use this credential as input for every VDP transfer transaction.

The VDP transfer is comprised of the following stages: The user retrieves its tokens from the ledger (2.1). The user contacts the data analyst (2.2) and executes the BindRandomness protocol. Thus, the user obtains a verifiable random value (2.3). The user executes the VDP transfer algorithm in three stages. First, the user adds noise to their attributes using the verifiable LDP mechanism (creates the DPA) and encrypts the result using the data analyst's public key. Then, the user computes the zk-SNARK proofs needed to verify the jointly generated randomness and the integrity of the DPA. Finally, the user uses the underlying transfer scheme in a black box manner to create a new unspent token. The resulting transfer transaction and the DPA encrypted with the analyst's public encryption key are posted on

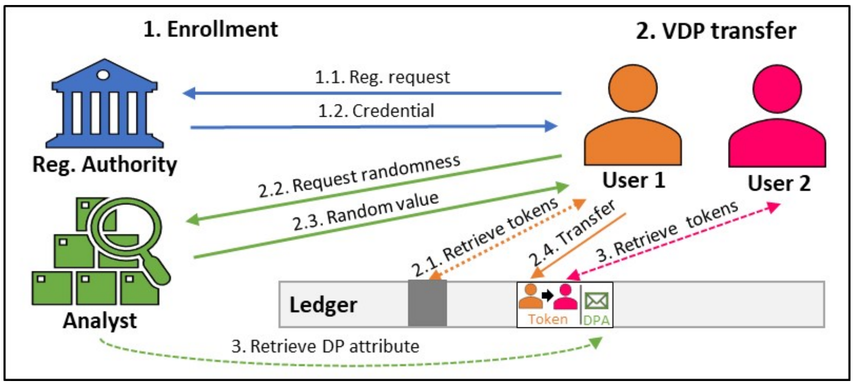


Figure 1 Overview of the VDP Transaction scheme.

the ledger (2.4). Finally, the analyst (3) decrypts the encrypted DPA and includes it in its statistical computation, and the second user (3) (i.e., the recipient of the transaction) can now use the newly created token.

4 The VDP Transaction Scheme

This section details the complete VDP transaction scheme.

4.1 The BindRandomness Protocol

The BindRandomness protocol presented in Figure 2 is a privacy-preserving verifiable protocol executed between the user and the analyst. This protocol has two main goals: (i) Obtaining an unbiased random value jointly generated by the user and the analyst. (ii) Computing a verifiable and unlinkable serial number. This serial number will enable the user to later prove in zero knowledge that they know a random value jointly generated with the analyst to be used as a source of randomness for a randomized algorithm.

The BindRandomness protocol takes as input a security parameter λ and a vector $\vec{\rho} = (\rho_1, \dots, \rho_m)$ s.t. $\forall i \in [m]\colon \rho_i \in \{0,1\}^\lambda$.
$\vec{\rho}$ represents m distinct seeds of unspent input tokens, and outputs two random values ξ_u, ξ_A, a commitment cm_{ξ_u}, and a signature σ_A.

In the initial stages of the protocol, the user samples a random value ξ_u, commits to it using $\mathsf{COMM}(\xi_u)$, and computes the serial number ν_1 with $\mathsf{PRF}_1(\vec{\rho})$. The user then sends the commitment cm_{ξ_u} and the serial number ν_1 to the analyst. The analyst checks if ν_1 was observed before and if so it aborts. Otherwise, the analyst will add ν_1 to some accumulator ACC, and continue on with sampling a random value ξ_A and signing $(cm_{\xi_u}, \nu_1, \xi_A)$ to obtain σ_A. The protocol ends with the analyst sending the user ξ_A and the signature σ_A. The user verifies the signature and discards ξ_A if the signature is found to be invalid.

After the protocol's execution, the analyst has a value ν_1 in its accumulator, which is correlated to m unspent tokens that the user may use in a future transaction. By signing the accompanied commitment of the user cm_{ξ_u} alongside ξ_A, the randomness picked by the user, as well as the randomness picked by the analyst are both indirectly bound to $\{\rho_i\}_{i=1}^m$ the seeds used for the unspent input tokens. As we will show in the next section, this is a crucial part of the security of our scheme: If the user used an unspent input's seed ρ_i to obtain randomness from the analyst, it must also use the corresponding unspent input in a transfer in order to use the randomness for the LDP computation.

BindRandomness

	User		Analyst
1 :	$\xi_u \leftarrow\!\$ \{0,1\}^\lambda$		
2 :	$cm_{\xi_u} = \mathsf{COMM}(\xi_u)$		
3 :	$\nu_1 = \mathsf{PRF}_1(\vec{\rho})$		
4 :		$\xrightarrow{(cm_{\xi_u}, \nu_1)}$	
5 :			Check $\nu_1 \notin \mathrm{ACC}$
6 :			$\mathrm{ACC} \leftarrow \mathrm{ACC} \cup \{\nu_1\}$
7 :			$\xi_A \leftarrow\!\$ \{0,1\}^\lambda$
8 :			$\sigma_A = \mathsf{Sign}(cm_{\xi_u}, \nu_1, \xi_A)$
9 :		$\xleftarrow{(\sigma_A, \xi_A)}$	
10 :	**return** $\xi_u, cm_{\xi_u}, \sigma_A, \xi_A$		

Figure 2 The BindRandomness protocol.

4.2 The VerRR Mechanism

The VerRR algorithm is a simple, verifiable LDP mechanism based on *randomized response* presented in Section 2.3. This mechanism makes one binary attribute of the user differentially private. In this setting, the user can answer any question that requires a binary answer. For example, to answer the question "Are you a non-profit?" the user can reply "0" for non-profit or "1" for pro-profit. To answer the question "Are you based in the U.S.?" the user can reply "0" for a yes or "1" for a no.

We chose this simple implementation as a proof of concept, but argue that this mechanism can be easily replaced by a more sophisticated LDP mechanism, such as one capable of handling histogram queries as in the *Rappor* mechanism used by Google [25]. As evident from our security proofs in Section 4.4, our technique achieves simulation security; therefore, the entire protocol inherits the security of the LDP function.

The VerRR algorithm uses the jointly generated random value ξ as the source of randomness needed to compute the double coin toss coin_1 and coin_2. Based on the results of the coin tosses, the algorithm determines the value of output $\widehat{\mathsf{dpa}}$ (the original dpa value, or a random output of "0" or "1").

The pseudocode for the VerRR algorithm is given in Algorithm 1.

Algorithm 1 VerRR.

Input:

- Verifiable randomness ξ
- A private attribute dpa

Output:

- Two coin toss results coin_1 and coin_2
- A differentially private value $\widehat{\mathsf{dpa}}$

```
1: Compute first coin toss
   $\mathsf{coin}_1 = (\xi \bmod 4) \bmod 2$
2: Compute second coin toss $\mathsf{coin}_2 = (\xi \bmod 4)/2$
3: if $\mathsf{coin}_1 = 0$ then
4:     $\widehat{\mathsf{dpa}} \leftarrow \mathsf{dpa}$
5: else
6:     if $\mathsf{coin}_2 = 0$ then
7:         $\widehat{\mathsf{dpa}} \leftarrow 1$
8:     else
9:         $\widehat{\mathsf{dpa}} \leftarrow 0$
10:    end if
11: end if
12: Output $\mathsf{coin}_1, \mathsf{coin}_2, \widehat{\mathsf{dpa}}$
```

Algorithm 2 VDPtransfer.

Input:
- Public parameters pp
- Verifiable randomness ξ_u and its commitment cm_{ξ_u}
- Serial number ν_1
- Verifiable randomness ξ_A
- Signature σ_A for $(cm_{\xi_u}, \nu_1, \xi_A)$
- Underlying transfer parameters: x=$((\theta_i^{old})_{i=1}^m, (\mathsf{addr}_{\mathsf{sk},i}^{old})_{i-1}^m, info$[7]$)$

Output:
- New tokens $(\theta_i^{new})_{i=1}^m$
- Encrypted VDP value δ
- Transfer transaction $\mathsf{tx}_{\mathsf{VDP}}$

```
1: Compute randomness $\xi = \mathsf{ADD}(\xi_u, \xi_A)$
2: Compute commitment $cm_\xi = \mathsf{COMM}(\xi)$
3: for each $i \in [m]$:
   a: Parse $\theta_i^{old}$ as $(\rho_i, \mathsf{dpa}, \mathsf{addr}_{\mathsf{pk},i}^{old}, *)$
   b: Parse $\mathsf{addr}_{\mathsf{sk},i}^{old}$ to retrieve $a_{\mathsf{sk},i}^{old}$
   c: Parse $\mathsf{addr}_{\mathsf{pk},i}^{old}$ to retrieve $a_{\mathsf{pk},i}^{old}$
4: Compute $\nu_2 = \mathsf{PRF}_2(\vec{\rho})$ $s.t.$ $\vec{\rho} := (\rho_1, \dots, \rho_m)$
5: Compute $\mathsf{VerRR}(\xi, \mathsf{dpa})$ to retrieve
   $(\mathsf{coin}_1, \mathsf{coin}_2, \hat{\mathsf{dpa}})$
6: Set $\vec{w_\xi} = (\xi_u, \xi_A, cm_{\xi_u}, \sigma_A, \xi, \vec{\rho}, \nu_1, (a_{sk,i}^{old})_{i=1}^m)$
7: Set $\vec{x_\xi} = (cm_\rho, \nu_2, (\mathsf{sn}_i^{old})_{i=1}^m, cm_\xi, \mathsf{pk}_A)$
8: Compute proof $\pi_\xi = \mathsf{Prove}(\mathsf{pk}_\xi, x_\xi, a_\xi)$
9: Encrypt and mask attribute
   $\delta = \mathsf{Enc}_{\mathsf{pk}_A}(r_u, \hat{\mathsf{dpa}})$
10: Set $\vec{w_\delta} = (\mathsf{coin}_1, \mathsf{coin}_2, \xi, \mathsf{dpa}, r_u, (a_{pk,i}^{old})_{i=1}^m)$
11: Set $\vec{x_\delta} = (\mathsf{COMM}_\xi, \delta, \mathsf{pk}_A, \mathsf{pk}_{RA})$
12: Compute proof $\pi_\delta = \mathsf{Prove}(\mathsf{pk}_\delta, x_\delta, a_\delta)$
13: Execute underlying transfer $\mathsf{Pour}(\mathrm{x})$
14: Set $\mathsf{tx}_{\mathsf{VDP}} = (\mathsf{tx}_{\mathsf{Pour}}, \delta, \pi_\xi, \pi_\delta)$
15: **Output** $(\theta_i^{new})_{i=1}^n$, $\mathsf{tx}_{\mathsf{VDP}}$
```

4.3 The VDP Transfer

The VDPtransfer algorithm expands the underlying transfer algorithm (e.g., the Pour algorithm used by Zcash [4]) by outputting additional information about the user's attributes. The analyst can later use this information to generate aggregate statistics regarding system users. To preserve privacy and allow plausible deniability, the user applies an LDP mechanism to its attributes, making them differentially private before disclosing them.

On a very high level, the algorithm expands the underlying transfer algorithm of unspent m tokens with randomness seeds $\vec{\rho}$ such that $|\vec{\rho}| = m$ executed by user u as follows:

1. u computes the jointly generated random value ξ out of the verifiable random values ξ_u and ξ_A obtained from executing the BindRandomness protocol.
2. u uses ξ as the source of randomness needed to generate the random values used in the VerRR algorithm.
3. u makes their attribute dpa differentially private and encrypts its value using the analyst's public key pk_A.
4. u computes two zero-knowledge proofs – the *Binding* proof π_ξ, and the *Encrypted VDP* proof π_δ – to prove the connection between the jointly generated random values ξ_u, ξ_A, ξ, the unspent tokens with random seeds $\vec{\rho}$, and the attribute dpa that was made differentially private. Additionally, u binds $(\rho)_{i=1}^m$ to $\nu_2 = PRF_2(\vec{\rho})$ and sends the underlying transfer encoding, the proofs ξ_u and ξ_A, and ν_2 to the analyst. Sending to the analyst ν_2 ensures that the user cannot reuse ξ a second time. The analyst is expected to add ν_2 to an accumulator ACC and ensure $\nu_2 \notin ACC$ upon receiving it from a user. Since ν_1 is computed using PRF_1 and ν_2 is computed using PRF_2, and $PRF_1 \neq PRF_2$ then also ν_1 is unlinkable to ν_2.

The pseudocode for the VDPtransfer algorithm is given in Algorithm 2.

[7] *info* represents the rest of the parameters needed for the underlying transfer

4.3.1 The *Binding* Proof

The proof, made by user u, is defined as follows:

$$\pi_\xi = \left[\begin{array}{l|l} \exists \xi_u, \exists \xi_A, & \mathsf{verify}(\mathsf{pk}_A, \sigma_A, ”cm_{\xi_u}||\nu_1||\xi_A”) = 1 \wedge cm_{\xi_u} = \mathsf{COMM}(\xi_u)\wedge \\ \exists cm_{\xi_u}, \exists \sigma_A, & \xi = \mathsf{ADD}(\xi_u, \xi_A) \wedge cm_\xi = \mathsf{COMM}(\xi) \wedge \nu_1 = \mathsf{PRF}_1(\vec{\rho})\wedge \\ \exists \xi, \exists \vec{\rho}, & \nu_2 = \mathsf{PRF}_2(\vec{\rho}) \wedge cm_\rho = \mathsf{COMM}(\vec{\rho}) \wedge \vec{\rho} \in \mathsf{ASC} \\ \exists \nu_1, \exists (a^{old}_{sk,i})^m_{i=1}) & \bigwedge^m_{i=1} \mathsf{sn}^{old}_i = \mathsf{PRF}_{a_{sk}}(\vec{\rho}[i]) \end{array} \right].$$

Where instances are of the form $\overrightarrow{x_\xi} = (cm_\xi, cm_\rho, \nu_2, (\mathsf{sn}^{old}_i)^m_{i=1}, \mathsf{pk}_A)$, and witnesses are of the form $\overrightarrow{w_\xi} = (\xi_u, \xi_A, cm_{\xi_u}, \sigma_A, \xi, \vec{\rho}, \nu_1, (a^{old}_{sk,i})^m_{i=1})$. We define ASC as an m relation $(n_1, n_2, ...n_m)|n_1 < n_2 < ... < n_m$ (i.e., an m relation where all the elements are smaller than the elements to their right).

An instance $\overrightarrow{x_\xi}$ specifies a commitment for a jointly generated randomness, a commitment for the unspent tokens, a public serial number binding the unspent tokens to the jointly generated randomness, the serial numbers of m distinct token (computed by the underlying token management system), and the analyst's public key. A witness $\overrightarrow{w_\xi}$ specifies user u's randomness and commitment to it, the analyst's randomness and signature for it, the jointly generated randomness, the seeds for the m distinct unspent tokens, the private serial number binding the unspent tokens to the jointly generated randomness, and the m private addresses of the m unspent tokens.

Given a *Binding* proof instance $\overrightarrow{x_\xi}$, a witness $\overrightarrow{w_\xi}$ is valid for $\overrightarrow{x_\xi}$ if the following statements hold:

1. The signature σ_A created by the analyst is valid, i.e., $\mathsf{verify}(\mathsf{pk}_A, \sigma_A, “cm_{\xi_u}||\nu_1||\xi''_A) = 1$
2. The commitment cm_ξ is a valid commitment for randomness ξ, i.e., $cm_\xi = \mathsf{COMM}(\xi)$.
3. The randomness ξ was computed using the user's randomness ξ_u and the analyst's randomness ξ_A, i.e., $\xi = \mathsf{ADD}(\xi_u, \xi_A)$.
4. The vector $\vec{\rho}$ is sorted in ascending order, i.e., $(\rho_1, \rho_2, ..., \rho_m)|\rho_1 < \rho_2 < ... < \rho_m$.
5. The serial numbers ν_1, ν_2 are computed correctly, i.e., $\nu_1 = \mathsf{PRF}_1(\vec{\rho})$ and $\nu_2 = \mathsf{PRF}_2(\vec{\rho})$.
6. The public serial number ν_2 matches a private serial number ν_1, s.t. ν_1 appears in the analyst's accumulator.
7. The commitment cm_u is a valid commitment for the randomness ξ_u generated by the user, i.e., $cm_{\xi_u} = \mathsf{COMM}(\xi_u)$.
8. The commitment cm_A is a valid commitment for the randomness ξ_A generated by the analyst, i.e., $cm_{\xi_A} = \mathsf{COMM}(\xi_A)$.
9. For each $i \in [m]$, the serial number sn^{old}_i of token θ^{old}_i is computed correctly, i.e., $\mathsf{sn}^{old}_i = \mathsf{PRF}_{a_{sk}}(\rho_i)$ s.t. $\rho_i = \vec{\rho}[i]$.

4.3.2 The *Encrypted VDP* Proof

The proof, made by user u, is defined as follows:

$$\pi_\delta = \left[\begin{array}{l|l} \exists \xi, \exists \mathsf{dpa}, & cm_\xi = \mathsf{COMM}(\xi) \wedge \hat{\mathsf{dpa}} = \mathsf{VerRR}(\xi, \mathsf{dpa})\wedge \\ \exists \mathsf{coin}_1, \exists \mathsf{coin}_2, & \mathsf{coin}_1 = (\xi \bmod 4) \bmod 2 \wedge \mathsf{coin}_2 = (\xi \bmod 4)/2\wedge \\ \exists r_u, \exists (a^{old}_{pk,i})^n_{i=1} & \delta = \mathsf{Enc}_{\mathsf{pk}_R}(r_u, \hat{\mathsf{dpa}}))\wedge \\ & \mathsf{verify}(\mathsf{pk}_R, \sigma_{\mathsf{dpa}}, “(a_{pk,i})^n_{i=1}||\mathsf{dpa}”) = 1 \end{array} \right].$$

Where instances are of the form $\overrightarrow{x_\delta} = (\mathsf{COMM}_\xi, \delta, \mathsf{pk}_A, \mathsf{pk}_{RA})$, and witnesses are of the form $\overrightarrow{w_\delta} = (\mathsf{coin}_1, \mathsf{coin}_2, \xi, \mathsf{dpa}, r_u, (a^{old}_{pk,i})^m_{i=1})$.

An instance $\overrightarrow{x_\delta}$ specifies a commitment for a jointly generated randomness, the encrypted value of the user's attribute after making it differentially private, the analyst's public key, and the registration authority's public key. A witness $\overrightarrow{w_\delta} = (\xi, \mathsf{coin}_1, \mathsf{coin}_2, \mathsf{dpa}, r_u, (a^{old}_{pk,i})^m_{i=1})$ consists of the jointly generated randomness and the two coin toss results derived from it, the user's attribute, the user's random scalar used during the ElGamal encryption process, and the m public addresses of the m unspent tokens.

Given an *Encrypted VDP* proof instance $\overrightarrow{x_\xi}$, a witness $\overrightarrow{w_\xi}$ is valid for $\overrightarrow{x_\xi}$ if the following statements hold:

1. The commitment cm_ξ is a valid commitment for randomness ξ, i.e., $cm_\xi = \mathsf{COMM}(\xi)$.
2. The double coin toss results coin_1 and coin_2 are derived from randomness ξ, i.e., $\mathsf{coin}_1 = (\xi \bmod 4) \bmod 2$ and $\mathsf{coin}_2 = (\xi \bmod 4)/2$.
3. The encrypted value δ was computed by encrypting the user's attribute dpa with the analyst's public key pk_A after making it differentially private using coin_1 and coin_2, i.e., $\delta = \mathsf{Enc}_{\mathsf{pk}_R}(\mathsf{LDP}(\mathsf{dpa}, \mathsf{coin}_1, \mathsf{coin}_2))$.
4. The signature created by the registration authority is valid, i.e., $\mathsf{verify}(\mathsf{pk}_R, \sigma_{\mathsf{dpa}}, \text{“}(a_{pk,i})^n_{i=1} || \mathsf{dpa}\text{”}) = 1$.

4.4 Security Analysis

4.4.1 Preserving Integrity

In our scheme, the user sends the analyst an encrypted result of computing the VerRR algorithm on their private attribute termed dpa. As the analyst does not see how the user computes the noise, it is crucial that the scheme preserves the integrity of the process in case the user is being dishonest. We prove a stronger result; If either (but not both) of the parties is malicious, the final result of our protocol distributes as an ideal functionality $\mathcal{F}$ for a randomized response differential privacy algorithm. In order to prove integrity, we first define three lemmas:

▶ **Lemma 3.** *Let $\mathcal{U}$ be the uniform distribution. Unless both*[8] *the user and the analyst are malicious, and if the analyst accepts the proofs accompanying the transaction,* BindRandomness *(see Figure 2) outputs ξ_A and ξ_u such that $\xi_A + \xi_u \sim \mathcal{U}\left(0, 2^\lambda\right)$.*

▶ **Lemma 4.** *Denote π_ξ and π_δ to be Non-Interactive Zero-Knowledge proofs accepted by the analyst with a corresponding commitment $COMM_\xi$ to ξ and δ being the claimed result*[9] *of applying VerRR on* $\mathsf{dpa} \in \{0,1\}^*$ *with some randomness ξ'. Then it holds that: $\xi = \xi'$.*

▶ **Lemma 5.** *Denote* dpa *to be the user's attribute and δ the result (claimed by the user) of* VerRR *on some* dpa' *accepted by the analyst. Then, it holds that indeed* $\mathsf{dpa}' = \mathsf{dpa}$.

The proofs can be found in section B in the appendix of our full version [18]. Now that we have proved the lemmas above, we can prove the integrity theorem (proof found in section B in the appendix of our full version [18]):

▶ **Theorem 6** (preserving integrity). *Let $\mathcal{F}$ an ideal functionality for the* VerRR *computation where the user (U) sends its data* dpa *and the analyst (A) receives* $\mathcal{F}(\mathsf{dpa})$*, and denote* $\langle U(\mathsf{dpa}), A\rangle$ *as a random variable that represents the final LDP value to be sent to the analyst. Then, unless both the user and analyst are malicious, it holds that:* $\langle U(\mathsf{dpa}), A\rangle \sim \mathcal{F}(\mathsf{dpa})$.

[8] If both are malicious, we cannot guarantee anything about the distribution of the final randomness $\xi = \xi_A + \xi_u$ as ξ_A, ξ_u may both not be sampled uniformly.

[9] In the protocol, δ is an encryption of the result, but for simplicity, we omit this.

Although Theorem 6 and its auxiliary lemmas are enough to prove the user cannot falsify $\mathcal{F}(dpa)$, a malicious user may attempt to execute several instances of the **BindRandomness** protocol and pick one result over the other in order to skew the randomness in its favor. We, therefore, prove that our scheme has a *finality* property with respect to the execution of **BindRandomness**:

▶ **Theorem 7** (Finality). *Let $\{sn_i\}_{i=1}^m$ be a set of serial numbers exposed by the underlying payment scheme, which correspond to tokens being spent in a future transaction by a user, and let $\vec{\rho}$ be its corresponding vector of random seeds. Denote $\{(\xi_u^{(i)}, \vec{\rho}) \mid \xi_u \in \{0,1\}^\lambda\}_{i=1}^\infty$ to be an infinite series of inputs to **BindRandomness** with the aforementioned $\vec{\rho}$ but with a different randomness $\xi_u^{(i)}$ each time. Then, only the first element in the series, $(\xi_u^{(1)}, \vec{\rho})$ grants the user with an output from the Analyst.*

The proof for Theorem 7 can be found in section B in the appendix of our full version [18]. Note that the binding proof π_ξ uses as input the serial numbers $\{sn_i\}_{i=1}^m$ and ensures that the input seeds for the serial numbers $\vec{\rho}$ are the same as the input seeds for ν_1 and ν_2, therefore a user cannot pick ν_1' for which $\nu_1' \neq PRF_1(\vec{\rho})$ in **BindRandomness** and then spend tokens corresponding to $\{sn_i\}_{i=1}^m$.

4.4.2 Preserving User Privacy

We show that our scheme does not degrade the privacy of the underlying transfer scheme which is used in a black box manner.

▶ **Theorem 8** (preserving privacy). *Let U be an honest user that completed a* VDPtransfer *preceded by interacting with the analyst via* BindRandomness*. Once the transfer completes, the analyst's guess about which user sent the* VDPtransfer *or interacted via* BindRandomness *is the same as it was before.*

In section B in the appendix of our full version [18], our proof for the theorem shows that the zero-knowledge proofs used in our scheme are simulatable, and hence anything computable by the analyst after observing them could have been computed before observing them.

5 Evaluation and Implementation

We implement our scheme in $\sim$ 500 lines of Go using the Gnark ZK-SNARK library [8]. Our implementation is available publicly in [17]. We use the Groth16 [27] scheme instantiated with the BN-254 curve as it is the most efficient according to the evaluation of [21].

Our choices of public key encryption, digital signatures, commitment schemes, and Pseudo-Random Functions were influenced by efficiency considerations, particularly the cost of operations in arithmetic circuits over a finite field that is the order of the BN-254 curve.

We use the MiMC [1] hash function for the Pseudo-Random Function and the commitment scheme. Although it gives only computational hiding and not information-theoretic hiding such as the Pedersen [39] commitment, it is cheaper because of the smaller[10] number of constraints.

For Public Key Encryption and digital signatures, we use Elgamal and edDSA, respectively, over a curve tailored to be efficient for ZK-SNARKs as it is defined over a field whose order [9] is the order of the BN-254 curve.

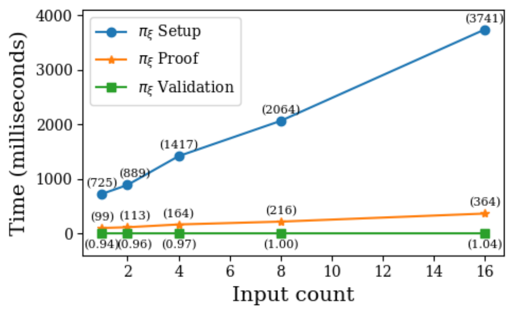
[10] Scalar multiplication in Elliptic Curves has logarithmic complexity.

Table 1 Performance of π_ξ proof. Times are in milliseconds.

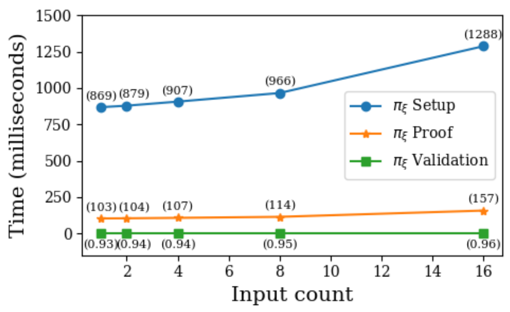
Inputs	Constraints	Setup	Proof	Ver
1	9769	725	99	0.944
2	12678	889	113	0.959
4	18496	1417	164	0.971
8	30132	2064	216	1
16	53404	3741	364	1.04

Table 2 Performance of π_δ proof. Times are in milliseconds.

Inputs	Constraints	Setup	Proof	Ver
1	12882	869	103	0.93
2	13155	879	104	0.935
4	13701	907	107	0.941
8	14793	966	114	0.951
16	16977	1288	157	0.961



(a) π_ξ Proof.



(b) π_δ Proof.

Figure 3 Performance Evaluation of both proofs as a function of the number of unspent inputs used in each transaction.

We investigate the overhead of using our scheme in conjunction with a privacy-preserving asset transfer by evaluating the time our proofs require.

We benchmark on a c5a.2xlarge AWS machine equipped with 8 vCPUs and 16GB RAM. We evaluate the performance of both our proofs (π_ξ, π_δ) by running 100 independent trials and computing the number of constraints, averages for setup time, proof time, and verification time for different numbers of input tokens. The results are depicted in Table 1 and Table 2 for π_ξ and π_δ respectively.

As seen from the performance evaluation results shown in Figure 3, our scheme has a practical execution time. Moreover, the number of inputs adds a negligible increase to the verification time, and has a small magnitude on the proof generation time. Additionally, as the verification time is very short (totaling less than 2ms for all input counts up to 16 inputs), we conclude that the additional computations and data that we added with our VDPtransfer transaction do not add much overhead to the original underlying transfer.

6 Discussion

6.1 The LDP Mechanism

Embedding any local differential privacy mechanism, especially a verifiable one, into a privacy-preserving transaction system, is a significant challenge. This is due to the need to maintain user privacy and prevent the unintentional leakage of information during data disclosure. For instance, applying the well-known *Laplace mechanism* [23] to render transaction values differentially private is ineffective when dealing with outliers, such as exceptionally high transaction values [40]. Merely adding noise following a Laplace distribution would not sufficiently conceal these extreme values, thereby compromising the privacy of the user involved in such a transaction. This approach, thus, does not align with our context. Given these observations, our focus has shifted to data requests that necessitate binary responses, which are inherently less susceptible to outlier effects.

For our local differential privacy mechanism, we utilized the randomized response technique [24]. We chose this mechanism for several reasons. Firstly, the randomized response method is based on simple probabilities, making it relatively straightforward to understand and analyze [44]. This characteristic is crucial when deploying the mechanism in a distributed, trustless environment. Secondly, the randomized response is well-suited for handling discrete values, aligning with our scheme's binary attribute. Lastly, the randomized response technique is suitable for local differential privacy and demonstrates strong accuracy and low error bounds, particularly when applied to many users [7]. These attributes make it applicable and well-suited to address the threat models specific to our research.

The challenge in a privacy-preserving transaction system was achieving verifiable randomness without revealing the user's identity. To the best of our knowledge, our approach is pioneering in addressing this issue. We devised a non-interactive method that employs two serial numbers for verification. While the user generates these serial numbers, the data analyst cannot link specific randomness to a particular transaction due to the unique creation and utilization of these numbers. Specifically, the first serial number is exclusively generated and applied during the randomness acquisition, and the second is solely utilized during the transfer as a component of the zk-SNARK proof, as detailed in Section 4.

6.2 The Zero-Knowledge Proof Scheme

Our approach uses the ZK-SNARK scheme of Groth16 [27], necessitating a trusted setup. A Zero-Knowledge proof system with a trusted setup stipulates that the randomness used during the setup process must be discarded to prevent misuse. If this randomness were to leak, any party possessing it could generate a deceptive proof. Contrarily, in a transparent setup, the randomness is publicly accessible and known to all parties, including the verifier, eliminating the need for an external trusted setup phase. Although it may seem that the transparent setup might be superior, in our specific context, a trusted setup aligns perfectly with our adversarial model, negating the need for a proof scheme incorporating a transparent setup.

The trusted setup is viable for us since our model consists of a single analyst and multiple (potentially infinite) users, where the analyst solely serves as a verifier, and the users operate as provers. Therefore, the analyst can generate the trusted setup since there is no other party that has to verify the proof generated by the analyst. Furthermore, since the data a user sends comprises cryptographic commitments, the analyst cannot derive any information from what a user sends, even if the randomness used during the trusted setup was not discarded.

6.3 Incentivising Conformation by Design

Our scheme has two phases: (a) obtaining randomness; (b) using it in a transaction. When obtaining the randomness to be later used in a transaction, the randomness is only bound to the unspent outputs that the user wishes to spend in the future. Consequently, if the user decides to use an unspent output for a transaction to some recipient, they may change their mind about the recipient but not about the (unspent) input tokens.

At first glance, it may seem like a dishonest user may want to deliberately skew the analyst's statistics by consistently selecting noise that hides their data. Such a dishonest user may obtain randomness for an unspent output, compute the corresponding noise, and if the noise is “bad”, they have three strategies: (i) Throw away the randomness and never use it; (ii) Use the randomness by sending its corresponding unspent outputs back to themselves and then repeating the process by obtaining new randomness; (iii) Trying to manipulate the randomness by requesting new randomness corresponding to different subsets of unspent outputs.

Discarding the randomness, as suggested in strategy (i), also means discarding the funds associated with the unspent outputs used to generate the randomness, which incurs a significant cost. Hence, there is an incentive for a dishonest user not to do so.

Surprisingly, using randomness by sending the unspent outputs back to the user, as suggested in strategy (ii), does not affect the analyst, as the analyst cannot differentiate between a transfer to the same user and a transfer to a different user in the first place. In other words, the analyst’s aggregated statistics stay the same whether dishonest users pick this strategy or not.

This leaves us with the dishonest user’s last strategy (i.e., strategy (iii)), binding various combinations of unspent outputs with the same total sum for one of the combinations to yield noise that hides the user’s data. In our implementation, the user cannot simply reorder the set of unspent inputs and request a new randomness since, as part of the π_ξ proof, the user proves that the random seeds of the unspent inputs are sorted in ascending order. Therefore, the only reason strategy (iii) is possible is that in our implementation, BindRandomness computes ν_1 as a PRF on a *vector* $\omega = (\rho_1, \rho_2, ..., \rho_m)$. Indeed, if the user has unspent outputs corresponding to $\{\rho_1, \rho_2, ..., \rho_l\}$ such that $l > m$, it has $\binom{l}{m}$ independent attempts of obtaining a randomness it desires. However, such a strategy can be easily mitigated by defining ν_1 as a *vector* instead of a single value. Specifically, if in BindRandomness the user sends: $\vec{\nu_1} = (PRF_1(\rho_1), PRF_1(\rho_2), ..., PRF_1(\rho_m))$ and the analyst checks $\nu_{1,i} \notin ACC$ for every $\nu_{1,i} \in \vec{\nu_1}$. This strategy becomes equivalent to the aforementioned strategy (i). We note that defining ν_1 as a *vector* instead of a single value also eliminates the need for the set of unspent inputs to be in ascending order.

7 Conclusions

In this work, we describe the VDP transaction scheme that fits the needs of digital payment systems that require built-in governance and regulations such as CBDCs. The scheme combines privacy-preserving transactions with statistical insights gathering without harming privacy. Since the VDP transaction scheme expands the functionality of any given privacy-preserving transaction system, it can uphold privacy guarantees towards the users. At the same time, since the VDP transaction scheme incorporates a mechanism for verifiable LDP, it can provide users with plausible deniability and prevent bias in users’ responses, thus maintaining the integrity of the statistical insights. To achieve verifiability, we adapt the implementation of the *random response* mechanism; We replace the randomness used in the original *random response* with a jointly generated randomness and add zk-SNARK proofs. Furthermore, we prove that our scheme can preserve user privacy and statistical insight integrity even if one of the main participants (i.e., the user or the analyst) is malicious.

References

1 Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. Cryptology ePrint Archive, Paper 2016/492, 2016. URL: https://eprint.iacr.org/2016/492.

2 Andris Ambainis, Markus Jakobsson, and Helger Lipmaa. Cryptographic randomized response techniques. In Feng Bao, Robert H. Deng, and Jianying Zhou, editors, *Public Key Cryptography - PKC 2004, 7th International Workshop on Theory and Practice in Public Key Cryptography, Singapore, March 1-4, 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 425–438. Springer, 2004. doi:10.1007/978-3-540-24632-9_31.

3 Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. Privacy-preserving auditable token payments in a permissioned blockchain system. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 255–267. ACM, 2020. doi:10.1145/3419614.3423259.

4 Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474. IEEE Computer Society, 2014. doi:10.1109/SP.2014.36.

5 Alex Biryukov and Sergei Tikhomirov. Deanonymization and linkability of cryptocurrency transactions based on network analysis. In *2019 IEEE European symposium on security and privacy (EuroS&P)*, pages 172–184. IEEE, 2019.

6 Ari Biswas and Graham Cormode. Verifiable differential privacy for when the curious become dishonest, 2022. arXiv:2208.09011.

7 Graeme Blair, Kosuke Imai, and Yang-Yang Zhou. Design and analysis of the randomized response technique. *Journal of the American Statistical Association*, 110(511):1304–1319, 2015.

8 Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. Consensys/gnark: v0.6.4, February 2022. doi:10.5281/zenodo.6093969.

9 Reinier Broker. *Constructing elliptic curves of prescribed order*. Leiden University, June 2006. Retrieved from https://hdl.handle.net/1887/4425.

10 Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 2020.

11 Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Data poisoning attacks to local differential privacy protocols. *CoRR*, abs/1911.02046, 2019. arXiv:1911.02046.

12 Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Data poisoning attacks to local differential privacy protocols. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 947–964, 2021.

13 Albert Cheu, Adam Smith, and Jonathan Ullman. Manipulation attacks in local differential privacy. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 883–900. IEEE, 2021.

14 Albert Cheu, Adam D. Smith, and Jonathan R. Ullman. Manipulation attacks in local differential privacy. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 883–900. IEEE, 2021. doi:10.1109/SP40001.2021.00001.

15 Christian Covington, Xi He, James Honaker, and Gautam Kamath. Unbiased statistical estimation and valid confidence intervals under differential privacy. *arXiv preprint arXiv:2110.14465*, 2021.

16 Ana-Maria Creţu, Federico Monti, Stefano Marrone, Xiaowen Dong, Michael Bronstein, and Yves-Alexandre de Montjoye. Interaction data are identifiable even across long periods of time. *Nature Communications*, 13(1):313, 2022.

17 Danielle Movsowitz Davidow, Yacov Manevich, and Eran Toch. ZKAT-VDP: Zero-Knowledge Asset Transfer - Verifiable Differential Privacy, 2022. URL: https://github.com/yacovm/ZKAT-VDP.

18 Danielle Movsowitz Davidow, Yacov Manevich, and Eran Toch. Privacy-preserving payment system with verifiable local differential privacy (full version). Cryptology ePrint Archive, Paper 2023/126, 2023. URL: https://eprint.iacr.org/2023/126.

19 Yves-Alexandre De Montjoye, Laura Radaelli, Vivek Kumar Singh, and Alex "Sandy" Pentland. Unique in the shopping mall: On the reidentifiability of credit card metadata. *Science*, 347(6221):536–539, 2015.

20 Maya Dotan, Saar Tochner, Aviv Zohar, and Yossi Gilad. Twilight: A differentially private payment channel network. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 555–570, 2022.

21 Guillaume Drevon and Aleksander Kampa. Benchmarking zero-knowledge proofs with isekai, 2019. URL: `https://sikoba.com/docs/SKOR_isekai_benchmarking_201912.pdf`.

22 Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 1–12, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

23 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

24 Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

25 Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

26 Stephen H Fuller and Ariel Markelevich. Should accountants care about blockchain? *Journal of Corporate Accounting & Finance*, 31(2):34–46, 2020.

27 Jens Groth. On the size of pairing-based non-interactive arguments. *IACR Cryptol. ePrint Arch.*, page 260, 2016. URL: `http://eprint.iacr.org/2016/260`.

28 Zhangshuang Guan, Zhiguo Wan, Yang Yang, Yan Zhou, and Butian Huang. Blockmaze: An efficient privacy-preserving account-model blockchain based on zk-snarks. *IEEE Transactions on Dependable and Secure Computing*, 19(3):1446–1463, 2022. `doi:10.1109/TDSC.2020.3025129`.

29 Muneeb Ul Hassan, Mubashir Husain Rehmani, and Jinjun Chen. Differential privacy in blockchain technology: A futuristic approach. *Journal of Parallel and Distributed Computing*, 145:50–74, 2020.

30 Muneeb Ul Hassan, Mubashir Husain Rehmani, and Jinjun Chen. Anomaly detection in blockchain networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 2022.

31 Wael Issa, Nour Moustafa, Benjamin Turnbull, Nasrin Sohrabi, and Zahir Tari. Blockchain-based federated learning for securing internet of things: A comprehensive survey. *ACM Computing Surveys*, 55(9):1–43, 2023.

32 Bin Jia, Xiaosong Zhang, Jiewen Liu, Yang Zhang, Ke Huang, and Yongquan Liang. Blockchain-enabled federated learning data protection aggregation scheme with differential privacy and homomorphic encryption in iiot. *IEEE Transactions on Industrial Informatics*, 18(6):4049–4058, 2021.

33 Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Extremal mechanisms for local differential privacy. *Advances in neural information processing systems*, 27, 2014.

34 Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

35 Fumiyuki Kato, Yang Cao, and Masatoshi Yoshikawa. Preventing manipulation attack in local differential privacy using verifiable randomization mechanism. In Ken Barker and Kambiz Ghazinour, editors, *Data and Applications Security and Privacy XXXV - 35th Annual IFIP WG 11.3 Conference, DBSec 2021, Calgary, Canada, July 19-20, 2021, Proceedings*, volume 12840 of *Lecture Notes in Computer Science*, pages 43–60. Springer, 2021. `doi:10.1007/978-3-030-81242-3_3`.

36 Gonzalo Munilla Garrido, Johannes Sedlmeir, and Matthias Babel. Towards verifiable differentially-private polling. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, pages 1–11, 2022.

37 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. URL: `https://bitcoin.org/bitcoin.pdf`.

38 Arjun Narayan, Ariel Feldman, Antonis Papadimitriou, and Andreas Haeberlen. Verifiable differential privacy. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, New York, NY, USA, 2015. Association for Computing Machinery. `doi:10.1145/2741948.2741978`.

39 Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

40 Rathindra Sarathy and Krishnamurty Muralidhar. Evaluating laplace noise addition to satisfy differential privacy for numeric data. *Trans. Data Priv.*, 4(1):1–17, 2011.

41 Georgia Tsaloli and Aikaterini Mitrokotsa. Differential privacy meets verifiable computation: Achieving strong privacy and integrity guarantees. In Mohammad S. Obaidat and Pierangela Samarati, editors, *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications, ICETE 2019 - Volume 2: SECRYPT, Prague, Czech Republic, July 26-28, 2019*, pages 425–430. SciTePress, 2019. `doi:10.5220/0007919404250430`.

42 Nicolas Van Saberhagen. Cryptonote v 2.0, 2013. Retrieved from `https://github.com/monero-project/research-lab/blob/master/whitepaper/whitepaper.pdf`.

43 Yu Wang, Gaopeng Gou, Chang Liu, Mingxin Cui, Zhen Li, and Gang Xiong. Survey of security supervision on blockchain from the perspective of technology. *Journal of Information Security and Applications*, 60:102859, 2021.

44 Yue Wang, Xintao Wu, and Donghui Hu. Using randomized response for differential privacy preserving data collection. In *EDBT/ICDT Workshops*, volume 1558, pages 0090–6778, 2016.

45 Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

46 Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

Correct Cryptocurrency ASIC Pricing: Are Miners Overpaying?

Aviv Yaish
The Hebrew University of Jerusalem, Israel

Aviv Zohar
The Hebrew University of Jerusalem, Israel

Abstract

Cryptocurrencies that are based on Proof-of-Work (PoW) often rely on special purpose hardware to perform so-called *mining* operations that secure the system, with miners receiving freshly minted tokens as a reward for their work. A notable example of such a cryptocurrency is Bitcoin, which is primarily mined using application specific integrated circuit (ASIC) based machines. Due to the supposed profitability of cryptocurrency mining, such hardware has been in great demand in recent years, in-spite of high associated costs like electricity.

In this work, we show that because mining rewards are given in the mined cryptocurrency, while expenses are usually paid in some fiat currency such as the United States Dollar (USD), cryptocurrency mining is in fact a bundle of *financial options*. When exercised, each option converts electricity to tokens.

We provide a method of pricing mining hardware based on this insight, and prove that any other price creates arbitrage. Our method shows that contrary to the popular belief that mining hardware is worth less if the cryptocurrency is highly volatile, the opposite effect is true: volatility *increases* value. Thus, if a coin's volatility decreases, some miners may leave, affecting security.

We compare the prices produced by our method to prices obtained from popular tools currently used by miners and show that the latter only consider the expected returns from mining, while neglecting to account for the inherent risk in mining, which is due to the high exchange-rate volatility of cryptocurrencies.

Finally, we show that the returns made from mining can be imitated by trading in bonds and coins, and create such imitating investment portfolios. Historically, realized revenues of these portfolios have *outperformed* mining, showing that indeed hardware is mispriced.

2012 ACM Subject Classification Applied computing → Digital cash; Security and privacy → Economics of security and privacy

Keywords and phrases Cryptocurrency, Blockchain, Proof of Work, Economics

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.2

Related Version See the full version of the paper for proofs and additional details.
Full Version: `https://doi.org/10.48550/arXiv.2002.11064` [90]

Funding This research was supported by the Ministry of Science & Technology, Israel.

1 Introduction

The cryptocurrency boom was heralded by the arrival of Bitcoin [56], which introduced the idea of a decentralized currency to the mainstream. Bitcoin relies on pseudonymous users called *miners* to operate the cryptocurrency's ledger in a decentralized manner. In particular, a computationally-heavy mechanism called PoW is used to achieve consensus between miners on the system's state and secure it from various attacks [72].

Miners are rewarded for their work via a form of computation based lottery, yielding additional rewards the more they compute on behalf of the system. These mining rewards have led to an arms-race in which miners purchase increasingly efficient and performant hardware [6]. Today's Bitcoin mining is mostly performed in industrial-scale mining "farms" hosting ASICs tailor-made for mining [61].

To stay competitive, miners buy mining rigs in advance at a significant capital expenditure, and they go to great lengths to keep their hardware's electricity cost at a minimum [61]. Thus, miners turn off their machines if it is unprofitable to mine [58, 68, 46, 42], and even transport hardware between remote locations on a seasonal basis to save on electricity [62, 18, 40].

The profits derived from mining are highly volatile as they depend on the erratic exchange-rate of the cryptocurrency received as reward (see Figure 1) and on the level of competition from other miners. These factors make mining a risky investment and may indirectly hurt the cryptocurrency if fewer miners are there to secure it.

Despite the high-risk returns from mining, mining calculators utilize basic techniques to evaluate the price of mining hardware. These naïve approaches revolve around a metric called the *hashprice*, which assumes that the currency's exchange-rate is *constant* and ignore the associated risk.

▶ **Definition 1** (Hashprice). *The* hashprice *of a specific mining machine is the expected profit that it produces per unit of computation, given that the exchange-rate of the mined token and the computational power mining it are constant until the end-of-life of the machine.*

This metric was introduced by the Luxur mining pool [47], and is widely used by the community [54, 67, 26, 60]. Indeed, this metric is used by the top 8 websites which correspond to the keywords *mining calculator* [81, 48, 53, 86, 29, 59, 21, 25, 25], according to the web traffic analyzer *similarweb* [70]. These sites were frequently recommended on mining-related resources [91, 74].

1.1 Our Approach

In contrast to using the expected rewards as captured by the hashprice, we advance a more nuanced approach for evaluating cryptocurrency mining hardware, such as ASICs that accounts for risk attitudes regarding the exchange rate. Risk attitudes are subjective, and hard to measure but are reflected in the exchange rate itself. We utilize tools from financial option pricing to incorporate the market's risk attitude into the ASIC pricing model.

Specifically, we show that using publicly available information about the market (such as the interest-rate) and information about the efficiency of hardware (such as hash rate and power consumption of each device), one can derive a *correct* price for the hardware. This price is correct in the sense that any other price creates *arbitrage* and thus allows market forces to earn a risk-free profit.

Then, we construct an investment portfolio of tokens and bonds which *imitates* an ASIC and provides the same profits as a given mining machine. This portfolio has the same price as the correct price of the machine which it imitates. We empirically evaluate such imitating portfolios over historical data and show that they earn more than the corresponding ASICs, while costing less.

To obtain the correct price and an imitating portfolio which has an equal cost, we prove ASICs are equivalent to a bundle of *financial options* that allow their owners to exchange electricity for coins at different points in time. Then, we present algorithms which compute the correct hardware cost and the corresponding imitating portfolio.

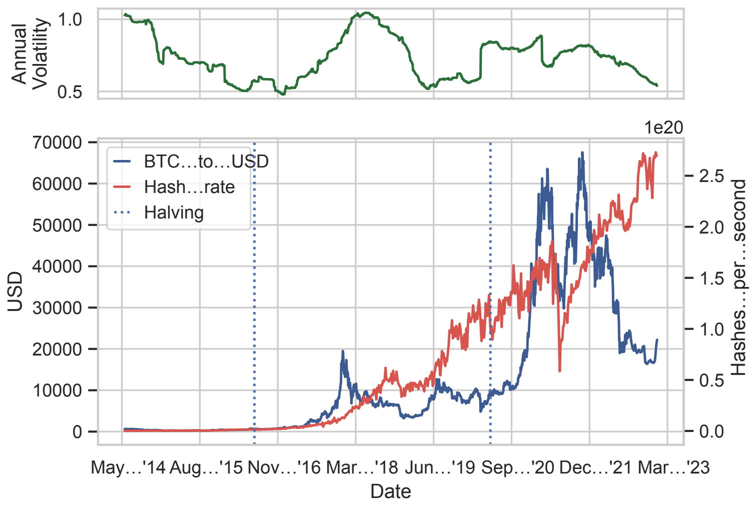


Figure 1 Bitcoin's exchange rate, annual volatility and global hash-rate, as functions of time.

Summary of Contributions

- **An economic model for PoW cryptocurrency mining.** We model mining while accounting for the inherent risk due to the volatile exchange-rate of cryptocurrencies. At first glance it may seem that higher volatility in rewards implies higher risk for miners, which may devalue mining machines, but in fact, we show that mining machines increase in value if the cryptocurrency is more volatile. This is because if the exchange rate plummets, the losses of miners are bounded (they can always shut off their machines and avoid paying for electricity), but if exchange rates increase steeply their gains can be significant.
- **An algorithm for pricing cryptocurrency mining hardware.** Using our model, we provide an algorithm that computes the price of an ASIC given its specifications and market parameters, without relying on subjective measures like a miner's risk preference or the expected exchange-rate of the mined cryptocurrency. We prove that any other price creates arbitrage.
- **The effects of risk and delay on the price of mining hardware.** We quantify the impact of the volatility of Bitcoin's exchange-rate and the delays miners frequently face when ordering new hardware on the value of mining machines.
- **Imitating portfolios for mining hardware.** We construct an *imitating portfolio* consisting of coins and bonds, which ideally provides the same returns as a specific ASIC.
- **Empirical evaluation.** We make a three-way comparison between historical hardware prices, the correct prices obtained by our results, and the costs of the corresponding imitating portfolios. Historically, our portfolios earn *more* than ASICs while costing less, even when considering trading fees. These results imply that ASICs are overpriced.

1.2 Motivating Example

To motivate our work, we show in Example 2 that the hashprice metric (given in Definition 1) is flawed. Furthermore, we show that more complex hardware pricing methods such as using the expected profit of a mining machine produce incorrect prices that create arbitrage opportunities.

▶ **Example 2.** A vendor offers the option of using its ASIC tomorrow to mine a single block. The vendor assures that if the ASIC is turned on, it will earn exactly 1 Bitcoin (henceforth denoted as BTC or ₿), and will require \$250 worth of electricity. For simplicity, let the interest rate be 1.

Assume bitcoin's value starts at \$400 today, and will either double or halve tomorrow with equal probability. Note that per the definition of the hashrate metric (see Definition 1), the price of the option is $\$400 - \$250 = \$150$. Furthermore, this price does not depend on the exchange-rate's random walk, but rather solely on its current value.

A more complex method to evaluate the price of the option would be to use its expected profits. At a \$200 rate, activating the ASIC will result in a loss of \$50, as \$250 is paid and only \$200 is received; thus, rational agents will not activate the ASIC, and will lose nothing. On the other hand, if the exchange rate increases to \$800, it is possible to earn $\$800 - \$250 = \$550$ by turning the hardware on. In total, the expected return is $\frac{1}{2} \cdot \$0 + \frac{1}{2} \cdot \$550 = \$275$.

It is tempting to say that this is the correct price for the option, but it does not take exchange-rate volatility into account. We later show that the correct price is in fact $\$183\frac{1}{3}$.

To show why both \$150 and \$275 are *incorrect*, note that these prices create arbitrage opportunities. We proceed by constructing a trading strategy that capitalizes on the arbitrage created by the latter price, and note that a similar strategy can be used for the former. Assume there is at least one rational buyer for the opportunity, willing to pay \$275. If so, that buyer will surely prefer purchasing it for the lower price of \$274!

We can sell the opportunity for the lower price *without* actually owning it, all the while promising the buyer that no matter the world state the same exact profits will be earned. Essentially, we are performing a short on the opportunity.

To fulfill the promise we do the following: immediately upon selling the opportunity we borrow $\$183\frac{1}{3}$ from the bank, giving us a total of $\$183\frac{1}{3} + \$274 = \$457\frac{1}{3}$. We buy ₿$\frac{11}{12}$, which under the current exchange-rate are worth $\frac{11}{12} \cdot \$400 = \$366\frac{2}{3}$. After this, we remain with a profit of $\$457\frac{1}{3} - \$366\frac{2}{3} = \$90\frac{2}{3}$, which we pocket as a profit. This is summarized in Table 1.

If bitcoin's value goes up, our rational buyer will want to turn on the (imaginary) ASIC and receive the promised ₿1 reward in exchange for the \$250 activation fee, which is paid to us. We use the fee to pay back the loan, leaving us with $\$250 - \$183\frac{1}{3} = \$66\frac{2}{3}$, exactly enough to buy ₿$\frac{1}{12}$, that together with our existing ₿$\frac{11}{12}$ can be given to the buyer as the mining reward, thus covering our short. Note we have also paid back all debt, while our pocketed $\$90\frac{2}{3}$ profit was untouched. The balances throughout the day are shown in Table 2.

On the other hand, if the value goes down, the rational buyer will not want to pay the activation fee as it is more expensive than the ₿1 (= \$200) profit; even if the buyer is interested in receiving a single bitcoin, buying it on the free market is cheaper than activating the ASIC. So, we have covered our short without having to pay the mining reward. We still need to repay our $\$183\frac{1}{3}$ debt, and luckily our coins are worth exactly $\frac{11}{12} \cdot \$200 = \$183\frac{1}{3}$. Again, we keep our pocketed profit. Table 3 presents all changes in our holdings.

Although we started with no money, we made a riskless profit of $\$90\frac{2}{3}$ due to the *incorrect* pricing of the ASIC. In Section 4, we show how to correctly price it, and prove that when using our method no arbitrage opportunities arise.

Organization

This paper is structured as follows: we present additional background on cryptocurrencies and option theory in Section 2. We go on to define a mining model in Section 3, and present our methods for correctly pricing ASICs in Section 4, deferring most proofs to the full version [90].We then employ our methods to perform an empirical evaluation using real-world data in Section 5. We go over related work in Section 6 and conclude with a discussion on the implication of our results and future work in Section 7.

Table 1 Balance of all assets on the first day of Example 2. In step #1, after selling the opportunity we have a -1 quantity of it, essentially performing a short on it.

#	Step	Cash	Debt	Coins	Opportunities
0	Start of day.	\$0	\$0	0	0
1	Sell opportunity.	\$274	\$0	0	-1
2	Borrow \$183$\frac{1}{3}$.	\$457$\frac{1}{3}$	\$183$\frac{1}{3}$	0	-1
3	Buy $\frac{11}{12}$ coins.	\$90$\frac{2}{3}$	\$183$\frac{1}{3}$	$\frac{11}{12}$	-1

Table 2 Balance of all assets on the second day of Example 2, if the exchange-rate has doubled. Regarding step #4: giving the buyer 1 coin covers the short on the opportunity.

#	Step	Cash	Debt	Coins	Opportunities
0	Start of day.	\$90$\frac{2}{3}$	\$183$\frac{1}{3}$	$\frac{11}{12}$	-1
1	Get activation fee.	\$340$\frac{2}{3}$	\$183$\frac{1}{3}$	$\frac{11}{12}$	-1
2	Pay loan back.	\$157$\frac{1}{3}$	\$0	$\frac{11}{12}$	-1
3	Buy $\frac{1}{12}$ coins.	\$90$\frac{2}{3}$	\$0	1	-1
4	Pay buyer 1 coin.	\$90$\frac{2}{3}$	\$0	0	0

Table 3 Asset balance on the second day, if the exchange-rate has halved. Rational buyers will not activate the ASIC for a loss, thus there is a 0 amount of the opportunity at step #0.

#	Step	Cash	Debt	Coins	Opportunities
0	Start of day.	\$90$\frac{2}{3}$	\$183$\frac{1}{3}$	$\frac{11}{12}$	0
1	Sell all coins.	\$274	\$183$\frac{1}{3}$	0	0
2	Pay loan back.	\$90$\frac{2}{3}$	\$0	0	0

2 Background

We now go over preliminary details necessary for our work. We begin by describing in Section 2.1 the mechanisms which underlie PoW cryptocurrencies, and by reviewing the economical considerations made by real-world miners in Section 2.2. We finish by giving a brief overview of option theory in Section 2.3.

2.1 Cryptocurrencies

Bitcoin and other similar tokens let users exchange funds by creating *transactions* [36] that are collected in *blocks* in a decentralized manner by pseudonymous users called *miners*, who are allowed to freely join or leave the system. The creation of blocks is called *mining*. To enforce some chronological order on transactions, each block must point to a preceding one, with the resulting data-structure often referred to as a *blockchain*. Thus, a blockchain is in essence a decentralized ledger of transactions, where blocks should ideally be mined one after the other.

Proof-of-Work

Bitcoin relies on a mechanism called PoW to ensure miners invest some expected amount of effort to create blocks, thus preventing miners from maliciously retroactively changing the ledger to their benefit [88]. This is enforced by requiring blocks to have a cryptographic hash [50] which is lower than some *target* value (when this hash is interpreted as a number). The hash function used in Bitcoin is *SHA256* [30]. Currently, the best known method for finding a low SHA256 hash is to try many different inputs by brute force [50]. Thus, the performance of mining hardware is measured by its *hash-rate*, the amount of hash calculations it can perform per unit of time. The mining target value is set by the mechanism to keep block creation rate roughly constant even when computational power is added to the network [89]. The specific mechanism overseeing this is called the *difficulty-adjustment algorithm (DAA)*. Thus, the probability that a single miner will create a block decreases if more hash-rate is competing against it.

In our work, we focus on Bitcoin. Historical data shows that Bitcoin's hash-rate was consistently more than 100 times higher than the combined power of other popular cryptocurrencies [8]. This allows us to avoid considerations such as "coin-hopping" (wherein miners switch between mining different cryptocurrencies), similarly to other papers [32, 89, 72, 37, 79]. Indeed, previous research shows that such behavior is rare in practice [51].

Mining Incentives

To encourage mining even in the face of the ever-mounting computational effort required, Bitcoin and similar cryptocurrencies reward the creator of each block with a *block reward*. As the size of blocks is limited, users can incentivize miners to prefer their transaction over others by paying a *transaction fee* to the first miner that includes it in a block. Transaction fees have roughly amounted to 1.5% of Bitcoin mining profits over the past year [16].

Single miners do not expect to find a block often, and so the majority of bitcoin mining is done in mining *pools* [85, 69, 76], where miners mine cooperatively and split rewards amongst themselves according to their relative contribution. Thus, small and constant returns can be expected by miners who take part in pools.

2.2 Real-World Considerations of Miners

Cryptocurrency exchange-rates and electricity costs are important considerations for miners [61, 33, 57, 84]. This is affirmed by large-scale miners, who claim to respond to market changes by turning mining rigs on and off "at a minute's notice" and "in real-time" [68, 46, 58]. Indeed, historical data indicates that miners rapidly turn hardware on and off, going as far as using old and inefficient hardware when the current rates deem it profitable [42]. On the other end of the spectrum, large-scale miners are not afraid of shutting hardware down for prolonged periods of time to move it to remote areas with cheap electricity [62, 18, 40].

Such behavior is facilitated by hardware and software vendors, who create products that are designed to rapidly switch between multiple low-power modes according to market conditions [10, 75, 42, 23, 22].

Even amateur miners use such optimizations by adopting after-market software that adds similar functionality to hardware which doesn't have it by default [74, 12, 3, 15, 82, 13, 83, 24].

2.3 Financial Options

A European *call-option* is a contract involving two parties and an underlying asset. By purchasing a call-option, the buyer receives from the seller the right to buy the asset at some agreed-upon price, the *strike price*, at a specific future date, the *expiration date*. As this is a right and not an obligation, the buyer need not exercise it if deemed unprofitable. For example, if by the date of expiry the underlying asset's price is lower than the strike price, it is preferable to buy the underlying asset directly and to discard the option.

In 1973, the Black-Scholes method for option valuation was proposed by [9], a seminal work in the field of option theory, and was later expanded upon by Merton [52]. Both rely on the *no-arbitrage* principle which argues that options should be priced such that no arbitrage possibility involving the underlying asset exists. Using option pricing as a foundation, various financial decisions have been cast as options [14, 78, 77, 27], for example the decision of whether to delay or abandon a project. This technique is called *real option valuation*.

Techniques from real option theory are introduced as needed throughout Section 4, with the required modifications for our setting. Further exploration of the topic is beyond the scope of this paper, but can be found in classic texts such as [19].

3 Model

We now describe an accurate model which accounts for the considerations made by real-world miners (see Section 2.2).

3.1 Mining Model

We divide time into discrete mining *opportunities* (or *turns*), and assumes a miner can either activate its hardware or leave it off for the whole duration of a single turn t.

If the ASIC has a hash-rate of h hashes-per-second and the total hash-rate active on the network excluding the ASIC is $H(t)$, activation of the ASIC allows the miner to receive a fraction $\frac{h}{H(t)+h}$ of the block-reward, which is R_t coins. This is a highly accurate approximation of the rewards earned by mining, as explained previously in Section 2.

Denote the ASIC's efficiency, measured in the Kilowatt-hours (kWhs) required for the computation of a mining opportunity, as φ, and the cost of electricity as e_t dollars per kWh.

To model hardware failures, we assume the ASIC "decays" gradually according to a mortality distribution: let $M(t)$ be the fraction of the ASIC that "remains" after t time units. For example, M can be a complementary cumulative density function (CDF) of some distribution [49]; let F be the CDF, then the complementary CDF is defined to be $1 - F$.

3.2 Financial Model

In our financial model of the world, for simplicity we call the mined cryptocurrency "Bitcoin", and refer to the fiat currency in which mining expenses are paid for as the USD, but both can be replaced by any other similar cryptocurrency and fiat currency.

We model the change in Bitcoin's exchange rate as a multiplicative random walk. We denote the Bitcoin-to-USD exchange rate at turn t by c_t, the probability for its value to rise to Δc_t in the next turn by q, and to fall to δc_t in the next turn by $1 - q$, resulting in a binomial price tree. A general form of such a tree is depicted in Figure 2.

While it may seem simplistic to assume that the price at every step can either increase or decrease by a factor, using sufficiently small steps yields a granular price model. Indeed, this distribution is commonly used in finance to model the value of assets such as currencies and stocks [14, 64]. Note that the length of each step of the exchange-rate's random walk does not have to coincide with the length of a mining opportunity. As we focus on evaluating a single mining opportunity, we use arbitrarily small steps to achieve a high granularity.

Denote the economy's annual multiplicative risk-fee rate as r. We assume $0 < \delta < 1 < r < \Delta$, otherwise, riskless arbitrage opportunities emerge, which we assume to not exist. This assumption is crystallized in Definition 3.

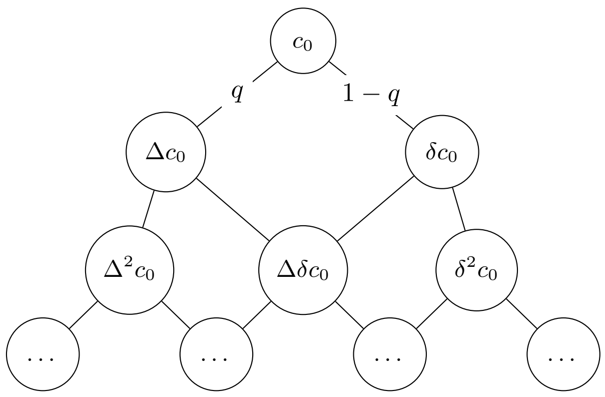


Figure 2 A coin's exchange rate as a multiplicative random walk, with a start value of c_0, a q probability to increase by a factor of Δ, and a $1-q$ probability to decrease by δ.

▶ **Definition 3** (The no-arbitrage assumption). *The free market adjusts asset prices such that it is impossible to outpace market gains without exposure to more risk. If such an arbitrage opportunity arises, market forces quickly use it until a pricing equilibrium is found, thus closing the opportunity.*

We mainly deal with the following types of assets:

i. The underlying cryptocurrency.
ii. A mining opportunity, denoting its value as $V(\cdot)$.
iii. A risk-free asset. An asset with a future return which is independent of the state of the world that is reached. Its multiplicative return is the *risk-free rate.* An example of such an asset is a government-issued bond, the value of which is denoted by B.

We also create portfolios holding combinations of the above assets, and denote their values by $\Phi(\cdot)$. We assume that assets are traded with sufficient liquidity, a clearly defined price and that it is possible to hold a "short" position on each one (owing the asset to another party, equivalent to holding a negative amount of it).

4 Theoretical Results

In this section, we derive the main results that allow us to evaluate the price of a mining machine, with all proofs given in the full version.

4.1 Pricing an ASIC

An ASIC gives its owner an option to activate it for each of the mining opportunities available during its lifetime, so an ASIC's value is exactly the sum of the values of all these opportunities, and by pricing a single opportunity we can price an ASIC.

▶ **Definition 4** (The value of the t-th mining opportunity, at turn k.). *Let $k \le t$. Given that the coin's exchange rate at k is c_k, we shall denote the value of the t-th opportunity at time k as $V(t,k,c_k)$.*

Some parameters (such as h) are left out of the notation for brevity.

Recall Example 2, which has demonstrated that it is hard to evaluate a future mining opportunity, e.g. calculate $V(t,k,c_k)$ when $k < t$, as the future exchange-rate is unknown. Specifically, that example examined a very basic case: $V(1,0,\$400)$. Thus, we take a step back and attempt to evaluate something easier, starting with each option's "immediate" value, which we soon define, and use a series of theorems and claims to evaluate a future option's value relative to arbitrary points in time, thereby giving the tools to calculate $V(t, k, c_k)$.

Total ASIC Value

Assuming we have successfully evaluated ASIC activation for a single turn, we can evaluate an "entire" ASIC received at time s, relative to time $t \leq s$:

$$V_{ASIC}(s, t, c_t) \overset{\text{def}}{=} \sum_{\tau=s}^{\infty} M(\tau - s) \cdot V(\tau, t, c_t) \tag{1}$$

Reception Delay

A method for evaluating ASIC prices could allow us to estimate the potential decrease in price associated with receiving hardware farther in the future. Often, ASIC manufacturers are backlogged and either deliver orders in the far future, or charge a premium for early deliveries. Assuming ASICs do not decay while in transit, the loss of receiving an ASIC at time s' instead of s is:

$$V_{ASIC}(s', t, c_t) - V_{ASIC}(s, t, c_t) \tag{2}$$

4.2 Pricing the Current Mining Opportunity

We begin by evaluating the t-th opportunity relative to turn t. Following Definition 4, this is notated by $V(t, t, c_t)$. At turn t, we know everything required to calculate the value of the t-th mining opportunity, as the biggest cause of uncertainty, the cryptocurrency's exchange rate c_t, is given. Thus, we call $V(t, t, c_t)$ the *immediate* value of the t-th mining opportunity.

Immediate Value of a Single Opportunity

At the t-th mining opportunity, an ASIC's owner has the option of paying the electricity cost of activating the ASIC for the duration of the opportunity, which under our model is $h \cdot \varphi \cdot e_t$, and in return receive the partial reward of $\frac{h}{H(t)+h} \cdot R_t \cdot c_t$. This opportunity can never be worth strictly less than zero, as a miner is not obliged to turn on its ASIC, and indeed a rational miner will not do so if it incurs a loss.

In total, the value at time t of the t-th mining opportunity is:

$$V(t, t, c_t) \overset{\text{def}}{=} \max\left(\frac{h}{H(t) + h} R_t c_t - h\varphi e_t, 0\right) \tag{3}$$

Shutdown Price

Immediately arising from Equation (3) is that if the cost of turning on the ASIC exceeds the profits, meaning that $\frac{h}{H(t)+h} R_t c_t \leq h\varphi e_t$, then no miner will turn it on, as paying the activation cost to buy the mined cryptocurrency on the free market is a better deal than actually using the hardware. This corresponds with the behavior of actual miners, as described in Section 2.2.

4.3 Pricing the Next Mining Opportunity

We now tackle the problem presented in the previous section more generally – pricing the t-th mining opportunity in relation to turn $t-1$. We do so by modifying techniques from option-pricing theory (as in [9, 28]). Specifically, to price this mining opportunity, we construct a portfolio of mining opportunities and coins at turn $t-1$.

The portfolio is crafted to yield identical valuations at turn t regardless of the change in the exchange-rate (see Claim 5). Thus, it is termed a *risk-free* portfolio. Its exact value at $t-1$ can be known by discounting and accounting for the risk-free rate (see Theorem 6).

We consider a portfolio that consists of the t-th mining opportunity and a short on (a yet to be chosen amount of) a_{t-1} coins, thus its value at turn $t-1$ is:

$$\Phi\left(t-1\right) = V\left(t, t-1, c_{t-1}\right) - a_{t-1}c_{t-1} \tag{4}$$

And, its value at turn t is:

$$\Phi\left(t\right) = V\left(t, t, c_t\right) - a_{t-1}c_t \tag{5}$$

▷ Claim 5. A portfolio holding the t'th mining opportunity and a short on a_{t-1} coins, where: $a_{t-1} = \frac{V(t,t,\Delta c_{t-1}) - V(t,t,\delta c_{t-1})}{c_{t-1}(\Delta-\delta)}$. is a risk free-portfolio for the turn between $t-1, t$. The portfolio's value in all possible states at t is: $\Phi\left(t\right) = V\left(t, t, \Delta c_{t-1}\right) - a_{t-1}\Delta c_{t-1}$.

The full version [90] contains a formal proof. The main idea is that there is one degree of freedom (choosing the short amount, a_{t-1}) which must satisfy an equation equating the value of the portfolio in both possible world states.

We now evaluate the return of the portfolio, and use it to price the mining opportunity.

▶ **Theorem 6.** *If no arbitrage opportunities exist, the multiplicative return of holding the portfolio constructed in Claim 5 between turns $t-1$ and t is equal to the risk-free rate.*

The proof (given in the full version [90]) shows that every other return contradicts the no-arbitrage assumption. As in Example 2, we can make a risk-free profit whenever such arbitrage opportunities arise.

We now reach an expression for the opportunity's price:

▶ **Corollary 7.** *The value of the t-th opportunity at $t-1$ is:*

$$V\left(t, t-1, c_{t-1}\right) = \frac{V\left(t, t, \Delta c_{t-1}\right) - V\left(t, t, \delta c_{t-1}\right)}{\Delta - \delta}\left(1 - \frac{\Delta}{r}\right) + \frac{V\left(t, t, \Delta c_{t-1}\right)}{r}$$

In the above, all factors can be calculated at time $t-1$.

We provide a proof in the full version [90]. It consists of using the return of the portfolio together with its values at turns $t-1$ and t to extract the value of the opportunity at $t-1$.

In Example 8, we revisit Example 2 and apply Corollary 7 to it.

▶ **Example 8.** Surprisingly, the price of the opportunity shown in Example 2 is lower than the naïve estimate. The opportunity's immediate value if the exchange-rate has gone up is:

$$V\left(1, 1, 800\right) = \max\left(1 \cdot 800 - 250, 0\right) = \$550$$

And, for the down state it is:

$$V\left(1, 1, 200\right) = \max\left(1 \cdot 200 - 250, 0\right) = \$0$$

By plugging the above into Corollary 7 we obtain the correct value of the opportunity at turn 0:

$$V\left(1,0,400\right)=\frac{550}{1}+\frac{550-0}{2-\frac{1}{2}}\left(1-\frac{2}{1}\right)=\$183\frac{1}{3}$$

According to Theorem 6, any other price creates arbitrage.

4.4 Pricing Relative to an Arbitrary Time

Algorithm 1 extends the previous method to evaluate the t-th opportunity relative to any previous point in time k. The idea behind the algorithm is to apply the same methods of Section 4.3 on every possible world-state, starting from turn t and going back, one step at a time, until reaching k. We now proceed to explain the method in depth.

Algorithm 1 MiningOpportunityValue.

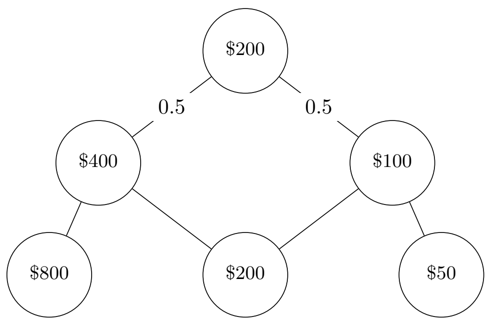
```
Input  : $t$ - the mining opportunity to evaluate.
         $k$ - the turn to evaluate relative to.
         $c_k$ - coin's exchange-rate at turn $k$.
Output : value of $t$-th opportunity at turn $k$.
for $c_t \in \{\Delta^{t-k}\cdot c_k, \Delta^{t-k-1}\cdot\delta\cdot c_k, \ldots, \delta^{t-k}\cdot c_k\}$ do
    $V\left(t,t,c_t\right) \leftarrow h\cdot\max\left(\frac{R_t\cdot c_t}{H(t)+h}-\varphi\cdot e_t, 0\right)$
end
for $\tau \in t-1,\ldots,k$ do
    for $c_\tau \in \{\Delta^\tau c_k, \Delta^{\tau-1}\delta c_k, \ldots, \Delta\delta^{\tau-1}c_k, \delta^\tau c_k\}$ do
        $a_\tau \leftarrow \frac{V(t,\tau+1,\Delta\cdot c_\tau)-V(t,\tau+1,\delta\cdot c_\tau)}{c_\tau\cdot(\Delta-\delta)}$
        $\Phi\left(\tau+1\right) \leftarrow V\left(t,\tau+1,\Delta\cdot c_\tau\right)-a_\tau\cdot\Delta\cdot c_\tau$
        $V\left(t,\tau,c_\tau\right) \leftarrow a_\tau\cdot c_\tau+\frac{\Phi(\tau+1)}{r}$
    end
end
return $V\left(t,k,c_k\right)$
```

The random-walk describing the coin's exchange rate for the period between turns k and t forms a tree with root c_k and leaves $\Delta^\tau\delta^{t-k-\tau}c_k$, for every $\tau\in[0,t-k]$. The leaves represent the trivial cases for evaluation, each one corresponds to a possible world state at turn t. As the opportunity expires at that turn, its value can be calculated directly from the definition given in Equation (3).

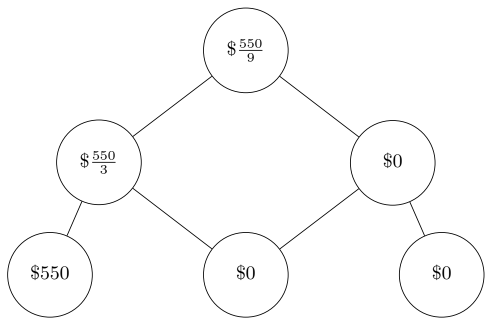
Proceeding inductively, let $\tau\in[k,t-1]$. We shall evaluate the opportunity at one of the vertices of the $(\tau-k)$-th level, assume it is c_τ. It points to two vertices from level $\tau-k+1$, specifically $\Delta c_\tau, \delta c_\tau$. Section 4.3 suggests that if the opportunity values for these two vertices are already calculated, the opportunity's value at c_τ's world-state can be obtained. Claim 9 covers this case.

▷ Claim 9. Let $\tau<t$. Given that the opportunity's valuations at $\tau+1$ are known, it is possible to evaluate $V\left(t,\tau,c_\tau\right)$, which is equal to:

$$\begin{aligned}V\left(t,\tau,c_\tau\right)=&\frac{V\left(t,\tau+1,\Delta c_\tau\right)-V\left(t,\tau+1,\delta c_\tau\right)}{\Delta-\delta}\left(1-\frac{\Delta}{r}\right)\\&+\frac{V\left(t,\tau+1,\Delta c_\tau\right)}{r}\end{aligned}$$



(a) Example 11's equiprobable two turn random walk, with a starting exchange-rate of $200 per BTC.



(b) The value of Example 11's mining opportunity at each possible world state, according to Algorithm 1.

Figure 3 Visual depictions of the possible states for Example 11's exchange-rate random walk, and the corresponding mining opportunity values and imitating portfolios.

The proof is given in the full version [90]. It uses the valuations at $\tau + 1$ to create a risk-free portfolio at turn τ that holds the t-th opportunity. The return of the portfolio at $\tau + 1$ can then be used to retrieve the value of the opportunity, similarly to Corollary 7.

By applying Claim 9 on every vertex of the current level and continuing in a dynamic manner to previous levels, it is possible to reach our goal and finally derive the value at the root of the tree, which corresponds to turn k.

A formula for a mining opportunity's value

Careful mathematical reasoning can be applied to Algorithm 1 to derive a formula for the value of the t-th opportunity:

▶ **Theorem 10.** *Let* $\gamma_{\downarrow} = \frac{1-\frac{\Delta}{r}}{\Delta-\delta}$, $\gamma_{\uparrow} = \gamma_{\downarrow} + \frac{1}{r}$, $\tau_0 = \left\lceil \frac{\log\left(\frac{(H(t)+h)\varphi e_t}{R_t \delta^{t-k} c_k}\right)}{\log\left(\frac{\Delta}{\delta}\right)} \right\rceil$. *The value of the t-th mining opportunity at turn* $k < t$ *is:*

$$V\left(t,k,c_k\right) = \sum_{\tau=\tau_0}^{t-k} \frac{\binom{t-k}{\tau}\gamma_{\uparrow}^{\tau}}{\left(-\gamma_{\downarrow}\right)^{k+\tau-t}} V\left(t,t,\Delta^{\tau}\delta^{t-k-\tau}c_k\right)$$

The proof is given in the full version [90]. By recursively applying Claim 9 on $V\left(t,k,c_k\right)$, a sum that only includes values of immediate opportunities is reached; this sum is shortened by ignoring opportunities with zero value. By Theorem 6, the value which is obtained is the only one which does not violate the no-arbitrage principle.

Example 11 shows how to use Theorem 10 in a complex setting.

▶ **Example 11.** Assume that bitcoin's exchange-rate at turn 0 is $200, and can either double or halve with equal probability. Extending the walk to two turns produces the tree in Figure 3a.

Furthermore, assume the vendor from Example 2 offers you the option of using its ASIC at the second turn for 10 minutes, under the same conditions as before. By following Algorithm 1, the value of the opportunity at each state can be calculated, as shown in Figure 3b. The algorithm proceeds as follows:

We start from the leaves and evaluate the immediate value of the opportunity at each one. At the leaf where the exchange-rate is \$800, the opportunity is worth \$550. On the other hand, if the rate is either \$200 or \$50, the opportunity is worth \$0. We have determined the value of the opportunity at all possible states of turn 2.

Now, by using Claim 9 on each of the two possible states at turn 1, we get that the value of the opportunity can be either $\$\frac{550}{3}$ (if the exchange rate is \$400) or \$0 (if it is \$100).

Finally, we take one step back and look at turn 0. By employing Claim 9 again together with our previous results, we find that the opportunity is worth $\$\frac{550}{9}$ at the first turn.

4.5 Imitating Portfolio

Buying mining hardware can entail difficulties: storing and maintaining it is costly, and receiving ordered ASICs promptly requires paying a hefty premium when demand is high.

Imitating an ASIC's revenue using purely financial means (e.g., an investment portfolio of tokens) might be better – it can start to produce revenue immediately without waiting, and avoids the aforementioned expenses. In Theorem 12, we show construct such a portfolio using coins and bonds.

▶ **Theorem 12.** *At turn τ, it is possible to construct an* imitating portfolio *for the t-th mining opportunity which is comprised of tokens and bonds. If this portfolio is properly adjusted at each turn until reaching time t, it can be sold to produce the same profits as the imitated mining opportunity.*

The proof relies on a series of claims, which we go over now. The portfolio we construct imitates the t-th opportunity between turns $\tau, \tau+1$, for $\tau < t$. Denote by $\overline{a}_\tau, B_\tau$ the respective amount of coins and risk-free bonds in the imitating portfolio at time τ. Thus, the portfolio's value at time τ is:

$$\overline{\Phi}(\tau) = B_\tau + \overline{a}_\tau \cdot c_\tau \tag{6}$$

And, at $\tau+1$ it is

$$\overline{\Phi}(\tau+1) = r \cdot B_\tau + \overline{a}_\tau \cdot c_{\tau+1} \tag{7}$$

▷ Claim 13. If there are no fees for trading bonds and coins, a portfolio can be constructed at turn τ to be worth exactly the same as the t-th mining opportunity in all world-states of turn $\tau+1$: $\overline{\Phi}(\tau+1) = V(t, \tau+1, c_{\tau+1})$. This portfolio is comprised of $\overline{a}_\tau$ tokens and B_τ risk-free bonds, where:

$$\overline{a}_\tau = \frac{V(t,\tau+1,\Delta\cdot c_\tau) - V(t,\tau+1,\delta\cdot c_\tau)}{c_\tau\cdot(\Delta-\delta)}$$
$$B_\tau = \frac{\Delta\cdot V(t,\tau+1,\delta\cdot c_\tau) - \delta\cdot V(t,\tau+1,\Delta\cdot c_\tau)}{r\cdot(\Delta-\delta)}$$

The proof is similar to that of Claim 5, see the full version [90] for details.

▷ Claim 14. At turn τ, the portfolio constructed in Claim 13 is equal in value to the t-th mining opportunity: $\overline{\Phi}(\tau) = V(t, \tau, c_\tau)$.

The proof is given in the full version [90]. It relies on showing that at turn τ the risk-free portfolio of Claim 9 is equal in value to B_τ. Finally, the claim is reached by applying algebraic manipulations to the definitions of the risk-free portfolio and the portfolio of Claim 13.

We finish the proof of Theorem 12 by combining Claims 13 and 14.

▶ **Corollary 15.** *The portfolio of Claim 13 is an imitating portfolio for the t-th mining opportunity between turns τ, $\tau+1$, meaning the portfolio is equal in value to the opportunity at both turns. Additionally, if there are no fees, selling the imitating portfolio for turns $\tau, \tau+1$ at turn $\tau+1$ generates enough money to buy the imitating portfolio for $\tau+1, \tau+2$. Thus, after the initial investment is made, no influx of funds is required to adjust the portfolio between turns, meaning that the initial purchase of the portfolio costs exactly the same as the opportunity that it imitates.*

Like in Section 4.4, the imitating portfolio can be evaluated at multiple time periods by dynamically moving backwards in time. Algorithm 2 provides an algorithmic construction of such a portfolio. If the portfolio changes between turns, the necessary adjustments cost additional fees; these are included in the empirical evaluation given in Section 5.

Algorithm 2 ImitateMiningOpportunity.

```
Input  : $t$ - the mining opportunity to imitate.
         $k$ - the turn to at which to create the portfolio.
         $c_k$ - coin's exchange-rate at turn $k$.
Output : an imitating portfolio for the $t$-th opportunity relative to turn $k$.
$\overline{a}_t \leftarrow 0$
$B_t \leftarrow 0$
for $c_t \in \{\Delta^{t-k} \cdot c_k, \Delta^{t-k-1} \cdot \delta \cdot c_k, \dots, \delta^{t-k} \cdot c_k\}$ do
    $V(t,t,c_t) \leftarrow h \cdot \max\left(\frac{R_t \cdot c_t}{H(t)+h} - \varphi \cdot e_t, 0\right)$
end
for $\tau \in t-1, \dots, k$ do
    for $c_\tau \in \{\Delta^\tau c_k, \Delta^{\tau-1}\delta c_k, \dots, \Delta\delta^{\tau-1} c_k, \delta^\tau c_k\}$ do
        $\overline{a}_\tau \leftarrow \frac{V(t,\tau+1,\Delta \cdot c_\tau) - V(t,\tau+1,\delta \cdot c_\tau)}{c_\tau \cdot (\Delta - \delta)}$
        $B_\tau \leftarrow \frac{\Delta \cdot V(t,\tau+1,\delta \cdot c_\tau) - \delta \cdot V(t,\tau+1,\Delta \cdot c_\tau)}{r \cdot (\Delta - \delta)}$
        $\overline{\Phi}(\tau) \leftarrow B_\tau + \overline{a}_\tau c_\tau$
        $V(t,\tau,c_\tau) \leftarrow \overline{\Phi}(\tau)$
    end
end
return $\{(\overline{a}_\tau, B_\tau) \mid \tau \in k, \dots, t\}$
```

In Example 16, we construct an imitating portfolio using the results of Section 4.5.

▶ **Example 16.** Recall Example 11, we revisit it and construct imitating portfolios for each of the example's states. These portfolios are summarized in Figure 4. Portfolios are comprised of holdings in coins and bonds, thus we represent them as tuples where the left item is the amount of coins, and the right one is the bonds' value in USD. Portfolios are sold on the last turn, so all final portfolios hold no assets. The portfolios are constructed like so.

First, evaluate the opportunity's price at all states. Next, apply Claim 13 on each possible state at turn 1. The imitating portfolio for the state where the exchange-rate equals \$400 is comprised of $\frac{550-0}{400 \cdot (2-0.5)} = \frac{11}{12}$ coins, and $\frac{2 \cdot 0 - 0.5 \cdot 550}{1 \cdot (2-0.5)} = -\$\frac{550}{3}$ worth of bonds. On the other hand, if the exchange-rate is \$100 then the portfolio has $\frac{0-0}{100 \cdot (2-0.5)} = 0$ coins and $\frac{2 \cdot 0 - 0.5 \cdot 0}{1 \cdot (2-0.5)} = 0$ bonds. Finally, the portfolio for the first state has $\frac{\frac{550}{3} - 0}{200(2-0.5)} = \frac{11}{18}$ coins and $\frac{2 \cdot 0 - \frac{1}{2} \cdot \frac{550}{3}}{1 \cdot (2-0.5)} = -\frac{550}{9}$ bonds.

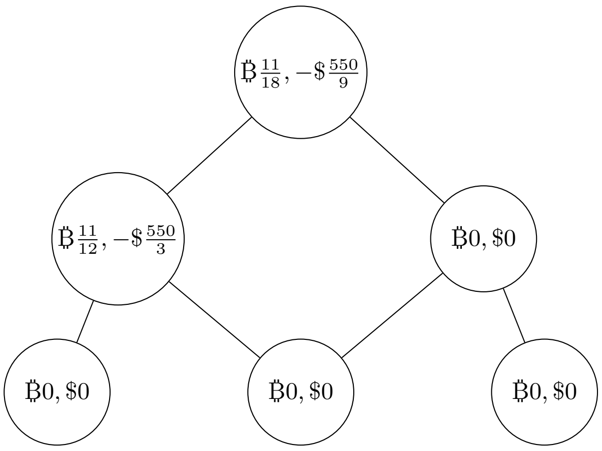

₿$\frac{11}{18}$, −$$\frac{550}{9}$
₿$\frac{11}{12}$, −$$\frac{550}{3}$
₿0, $0
₿0, $0
₿0, $0
₿0, $0

Figure 4 Imitating portfolios for each possible world-state of Example 16, per Algorithm 2.

To show that these portfolios are indeed imitating, we analyze their returns on the final turn. If an imitating portfolio is sold on the final turn, by construction its return should equal the one given by the actual mining opportunity.

If the exchange-rate is \$800, the portfolio we constructed is worth $800 \cdot \frac{11}{12} - \frac{550}{3} = \$550$, so selling it produces exactly the same profits as the opportunity at this state. If the exchange-rate is \$200, look at the two possible cases: if the previous turn's exchange-rate was \$400, our portfolio is comprised of $\frac{11}{12}$ coins and bonds worth $-\$\frac{550}{3}$, thus selling the portfolio gives a profit of $400 \cdot \frac{11}{12} - \frac{550}{3} = \$0$, again equal to the opportunity's. Conversely, if the previous rate was \$100, our portfolio holds no assets, so there is nothing to sell, and as before the profit is \$0, equal to the opportunity's.

5 Empirical Evaluation

We now employ our methods on real world data, deriving prices for the *Bitmain Antminer S9*, an ASIC which has dominated the market for an extended period of time, and constitutes around 33% of the currently active hash-rate on Bitcoin [42, 84].

5.1 Parameters

The parameters which are used throughout this section were obtained from real-world data, and were set to the following values:

ASIC price and specifications

We compare our prices to historical market prices which were obtained from the manufacturer's Amazon page. We took hardware specification from [80], and assumed ASICs last 2 years on average. In fact, hash-rate considerations imply that their profits vanish even faster.

Mining and imitation fees

In our evaluation, we compare between ASICs and their corresponding imitating portfolios. If there were multiple options for relevant parameters, we always chose the ones that make the imitating portfolios' job harder:

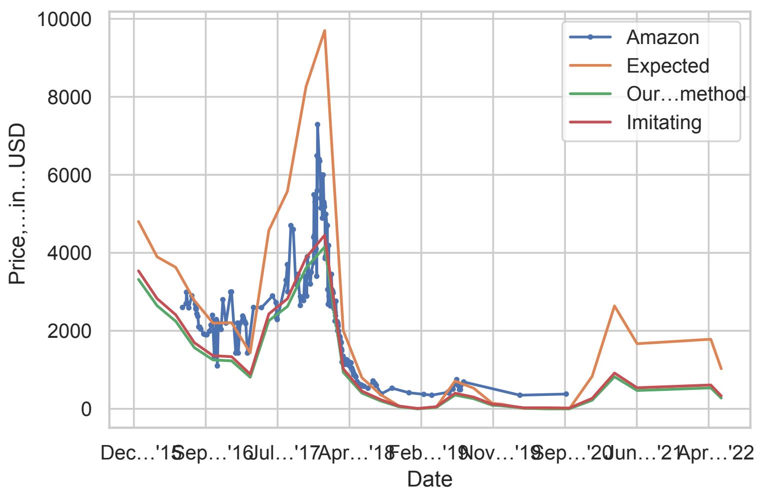


Figure 5 ASIC prices according to different valuation methods, as functions of time, including the costs associated with the corresponding imitating portoflios.

- *Electricity fees* were set to \$0.035 per kWh, lower than the average rates paid by industrial users and miners in the US [35, 1].
- *Mining pool fees* were set to 2%. Large pools (consistently comprising at least 40% of Bitcoin's hash-rate over the past year) have asked for 2.5% [38, 16, 87].
- *Trading fees* for bonds and BTC-to-USD were set to 1%, more than fees offered by large companies. For example, Coinbase asks for 0.6% at most [20].

Exchange rate, hashrate and interest rate

The historical BTC-to-USD exchange-rate and global hashrate were taken from `blockchain.com`. Annual volatility, defined as the standard deviation of log-returns, and future global hash-rate growth (which we assumed to be exponential in accordance with the literature [11]) were evaluated using data starting at 2013 and ending at the estimation date. The economy's annual risk-free rate was set to 2%.

5.2 Results

We now go over the results of our empirical evaluation.

Official Prices Do Not Account For Risk

We obtain the correct prices for the Antminer S9 by using Algorithm 1 with parameters corresponding to various points in time.

Figure 5 compares prices given by our method to Bitmain's official Amazon prices, and to a naïve evaluation method anecdotally used by miners (labeled "Expected"), which assumes the future BTC-USD exchange-rate will continue its recent rate of growth. This naïve method ignores risk and uses only expected values, as in Example 2. The official prices are closer to the naïve price, suggesting that *they do not fully account for risk.*

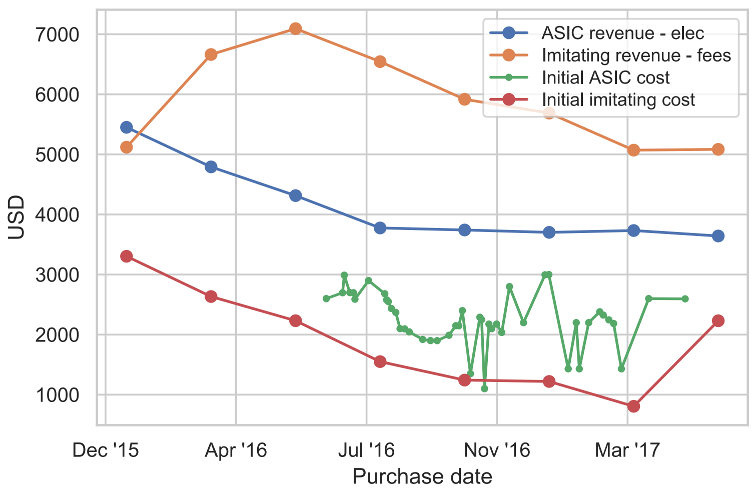
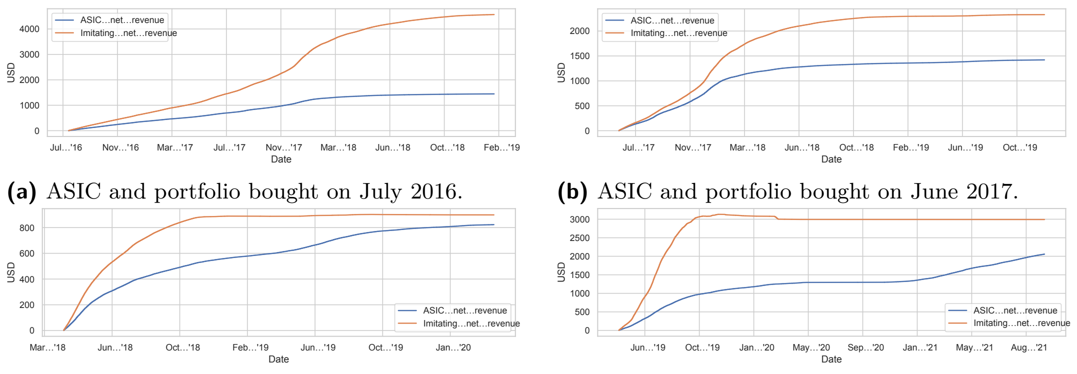


Figure 6 Realized revenue (after expenses) and initial cost for a 2-year operation of an ASIC and its corresponding imitating portfolio. An ASIC's initial cost is its Amazon price, and its expenses are the electricity it consumes. The portfolio's initial cost is the cost of buying it, and its expenses are the trading fees required for maintaining it over 2 years.



(a) ASIC and portfolio bought on July 2016.
(b) ASIC and portfolio bought on June 2017.
(c) ASIC and portfolio bought on April 2018.
(d) ASIC and portfolio bought on May 2019.

Figure 7 Realized revenue (after expenses) of an ASIC and its imitating portfolio, each bought for $1000 at different points in time.

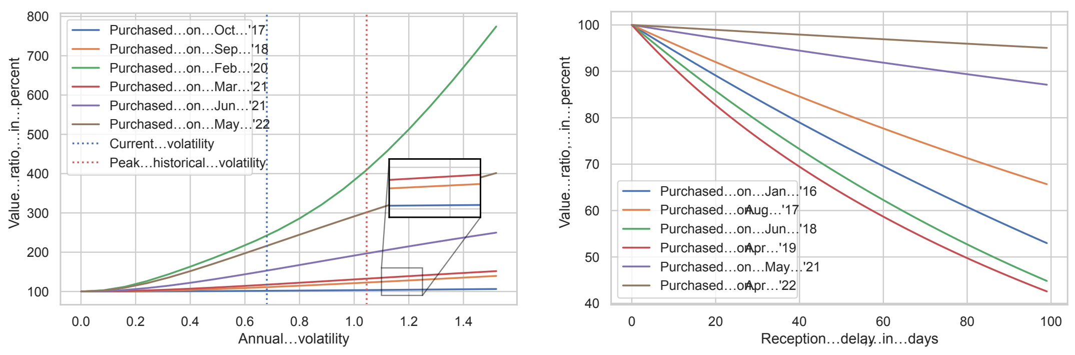
Imitating Portfolios

We now utilize Algorithm 2 to produce imitating portfolios for the Antminer S9. These portfolios are benchmarked and compared to the actual hardware using the realized exchange-rates and hash-rates to evaluated the returns made by each.

When evaluated on recent data, our imitating portfolios earned more than the equivalent ASICs, while costing less to buy and maintain, meaning ASICs are overpriced.

Figure 6 aggregates *realized* revenues and initial costs of ASICs and the corresponding imitating portfolios. We assume ASICs are received and activated *immediately* after purchase, which is far from typical as usually miners wait a long time to receive hardware. The revenue for both is *after* deducting all expenses (electricity for ASICs, and trading fees for portfolios).

Similarly, Figure 7 compares the realized revenue (after expenses) obtained from investing $1000 in an imitating portfolio with an equal investment in real mining hardware.



(a) Higher volatility increases ASIC value. **(b)** ASICs received later in time decrease in value.

Figure 8 The effects of volatility and delay on hardware value.

Our imitating portfolio's revenue is not equal to an ASIC's because of a gap between the realized and projected growth rates of the network's total hash-rate. Also, a portfolio's accuracy increases with the granularity of its time-steps, while the adjustments made at every step might increase its cost. We used 25 steps per mining opportunity, which empirically produces accurate results.

To provide an additional angle on these results, we show in Figure 5 both the correct prices of mining hardware over time, and the total cost of the corresponding imitating portfolios, including the average-case fees paid for all necessary adjustments. Although an imitating portfolio is more expensive when compared to the correct price, it still costs less than the official price.

Volatility Increases Value

Figure 8a depicts our evaluation of ASIC prices as a function of volatility, where each line represents a different purchase date. Bitcoin's annual volatility, as estimated on September 9th, 2021, and its peak annual volatility, which occurred in the year preceding April 29th, 2018, are depicted as vertical lines.

Our method gives higher prices for ASICs when volatility is higher. For example, an ASIC bought on June 2021 could cost 20% more if the volatility was at its historical peak.

Of note is the increase in value for hardware bought on February 2020. This can be explained by the crash in global hashrate experienced at the beginning of 2021 (see Figure 1). The combination of high volatility and low hashrate means that it is profitable to turn on hardware which might not be the most efficient or powerful (equivalently, the hardware's shutdown price is lower).

Reception Delay Decreases Value

By applying Equation (2) on historical data, we learn that a delay in the reception of an ASIC can severely lower its value, with a month's delay decreasing value by 30%, as seen in Figure 8b.

6 Related Work

To the best of our knowledge, our work is the first to evaluate the price of mining hardware, and to show that mining hardware can be imitated by purely financial means.

Economic Models of Mining

Other works have attempted to model the economics of mining without evaluating hardware prices, but most of these have not addressed the risk inherent in exchange-rate fluctuations and their affect on the economics of mining. For example [44, 45, 55], examine mining revenue in an economic setting where different cryptocurrencies co-exist. Several papers explore single-token economic models of mining, but most focus on the willingness of new miners to enter the market based on *expected* returns, and usually consider equilibria in a single shot interaction, e.g. [4, 31], or works such as [71], which consider a myopic Nash equilibrium in a game model of the bitcoin market. An equilibrium of miners in a bounded horizon setting is explored in [34, 39], both show that miners may gain by turning their hardware on and off repeatedly, thereby taking advantage of difficulty adjustments.

Some tried accounting for risk, for example [5], where the price of bitcoin (but not of mining hardware) is based on user adoption and friction due to exchange-rate uncertainty, or [7] which focuses on estimating hashrate allocation between multiple tokens by using miner risk-preference to estimate their expected revenue, which our method shows can give an incorrect result.

Concurrently and independently of our work, [43] consider mining hardware as an option, but present a simpler model that lacks several factors inherent to the mining market such as: changing electricity costs, hardware decay and delivery delays. Our work also adds empirical evaluation of the model compared to historical data and an analysis of the performance of imitating portfolios.

Economic Models of Cryptocurrency Security

An analysis of Bitcoin's security in a model where miner rewards are based on transaction fees and block-rewards are negligible is carried out in [79]. An economic analysis of the security of Bitcoin is performed by [17], arguing that when the currency is under attack, its value drops, causing mining hardware to lose value. In [88], it is shown that a malicious mining strategy strictly dominates the honest one in Ethereum-like cryptocurrencies, meaning that attacking the cryptocurrency is riskless when compared to the "honest" mining protocol, but can earn more profits.

Improving Mining Performance and Mining Pools

Some works attempted to improve the performance of mining machines [2, 41, 73], thus also increasing profits. But, these do not analyze the value of mining hardware.

A different approach is for so-called "solo" miners to operate as part of mining pools, which are coalitions of miners who mine together to get a steadier revenue-flow. Indeed, most mining is performed by pools [85]; thus, risk-aversion is believed to be widespread among miners. The economics of pools were examined by [63, 66, 65], which again neglected risk.

7 Conclusion

In this paper we show that widespread notions regarding ASIC prices and their dependence on subjective measures like projected expected exchange-rates are flawed. Instead, we present a method for *correctly* pricing mining hardware, and show ASICs can be *imitated* using bonds and tokens.

Popular opinion holds that as Bitcoin becomes more widely used, its volatility will decrease. Our evaluation shows that a decrease in volatility negatively affects the value of hardware, while at the same time making imitating portfolios cheaper to maintain (smaller adjustments are needed). Combined, both negate the financial incentives put in place to encourage mining. As Bitcoin's security relies on miner participation, lower mining revenues hurt security and undermine Bitcoin's usage as a currency.

Future Work

The security risk inherent in lower volatility can be addressed by adopting random reward mechanisms to artificially increase volatility: if rewards are made to follow a random walk, the returns of miners become more volatile, thus increasing potential profits and miner participation. To prevent miners from foreseeing future profits and stopping mining, rewards should be determined post-hoc.

We assumed that the global hash-rate is exogenous to the model, a possible extension could be to endogenize this. Miners may purchase hardware as long as it remains profitable to do so. Another interesting extension is to consider mining hardware capable of mining multiple currencies.

These additions could allow using our results to estimate the global-hash rate as dependent on the reward and difficulty adjustment mechanisms of a coin and its competitors, potentially helping to design better ones that avoid pitfalls like selfish-mining and "hash-wars". Hash-rate could also be analyzed in relation to a coin's exchange-rate, which are correlated according to anecdotal evidence, see Figure 1.

Glossary

Following is a list of important notations used in the paper.

V_{ASIC}	Value of an ASIC, in US dollars.
₿	The symbol for the Bitcoin cryptocurrency.
R	The reward received for mining a block, in tokens.
B	Government issued bonds, yielding the risk-free rate.
c	The value of a single coin, in USD.
δ	The multiplicative factor by which the coin's price can decrease.
$1-q$	The probability of a decrease in the coin's price.
φ	Kilowatt-hours required for the computation of a single mining opportunity.
e	Price of electricity, in US Dollars per kilowatt-hour.
H	The global hash-rate active on the network, in hashes-per-second.
h	The hash-rate of the ASIC to price, in hashes-per-second.
$\overline{a}$	Amount of coins to hold a long position on.
M	The ASIC's mortality distribution.
V	Value of a mining opportunity, in US dollars.
Φ	Value of a portfolio, in US dollars.
r	The yearly risk-free rate.
a	Amount of coins to hold a short position on.
Δ	The multiplicative factor by which the coin's price can increase.
q	The probability of an increase in the coin's price.

References

1 U.S. Energy Information Administration. Electric power monthly, 2022. URL: https://web.archive.org/web/20220818154159/https://www.eia.gov/electricity/monthly/epm_table_grapher.php?t=epmt_5_6_a.

2 J. Anish Dev. Bitcoin mining acceleration and performance quantification. In *2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–6, San Francisco, CA, USA, May 2014. IEEE. doi:10.1109/CCECE.2014.6900989.

3 Antpool. Antminertool manual, 2020. URL: https://web.archive.org/web/20201111172854/https://www.antpool.com/download/tools/002-BulkManagement-en.pdf.

4 Nick Arnosti and S. Matthew Weinberg. Bitcoin: A Natural Oligopoly. *Management Science*, 68(7):4755–4771, 2022. doi:10.1287/mnsc.2021.4095.

5 Susan Athey, Ivo Parashkevov, Vishnu Sarukkai, and Jing Xia. Bitcoin pricing, adoption, and usage: Theory and evidence, 2016.

6 M. Bedford Taylor. The evolution of bitcoin hardware. *Computer*, 50(9):58–66, 2017. doi:10.1109/MC.2017.3571056.

7 George Bissias, Brian N. Levine, and David Thibodeau. Using economic risk to model miner hash rate allocation in cryptocurrencies. In Joaquin Garcia-Alfaro, Jordi Herrera-Joancomartí, Giovanni Livraga, and Ruben Rios, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 155–172, Cham, 2018. Springer International Publishing.

8 BitInfoCharts. Bitcoin, ethereum, dogecoin, xrp, ethereum classic, litecoin, monero, bitcoin cash, zcash, bitcoin gold hashrate historical chart, 2022. URL: https://web.archive.org/web/20220522122528/https://bitinfocharts.com/comparison/hashrate-btc-eth-doge-xrp-etc-ltc-xmr-bch-zec-btg.html#3y.

9 Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973. doi:10.1086/260062.

10 Blockstream. Instant energy demand from the bitcoin network, 2021. URL: https://web.archive.org/web/20210824180952/https://blockstream.com/energy/.

11 R. Bowden, H. P. Keeler, A. E. Krzesinski, and P. G. Taylor. Modeling and analysis of block arrival times in the bitcoin blockchain. *Stochastic Models*, 36(4):602–637, 2020. doi:10.1080/15326349.2020.1786404.

12 braiins. Braiins os & braiins os+ custom asic firmware: optimize performance & efficiency, 2018. URL: https://web.archive.org/web/20210812132518/https://bitcointalk.org/index.php?topic=5036844.0.

13 BRAIINS. Autotuning mining firmware, 2022. URL: https://web.archive.org/web/20220425034423/https://braiins.com/os/plus.

14 Luiz E Brandão, James S Dyer, and Warren J Hahn. Using binomial decision trees to solve real-option valuation problems. *Decision Analysis*, 2(2):69–88, 2005. doi:10.1287/deca.1050.0040.

15 BTC.com. Using btc tools to do miners' batch management, 2019. URL: https://web.archive.org/web/20201129045355/https://help.pool.btc.com/hc/en-us/articles/360020105012-Miners-Batch-Management.

16 BTC.com. Pool stats, 2022. URL: https://web.archive.org/web/20220820062646/https://btc.com/stats/pool?pool_mode=year.

17 Eric Budish. The economic limits of bitcoin and the blockchain. Working Paper 24717, National Bureau of Economic Research, June 2018. doi:10.3386/w24717.

18 Scott Chipolina. Bitcoin's hash rate drops as china's rainy season ends, 2020. URL: https://web.archive.org/web/20211117210216/https://decrypt.co/46601/bitcoin-hash-rate-drop-attributed-to-chinese-rainy-season.

19 John H Cochrane. *Asset pricing: Revised edition*. Princeton university press, Princeton, NJ, USA, 2009.

20 Coinbase. What are the fees on coinbase pro?, 2022. URL: https://web.archive.org/web/20220406132137/https://help.coinbase.com/en/pro/trading-and-funding/trading-rules-and-fees/fees.

21 CoinWarz. Bitcoin mining calculator, 2022. URL: https://web.archive.org/web/20220514110643/https://www.coinwarz.com/mining/bitcoin/calculator.

22 Bitmain Technologies Holding Company. S19 server installation guide, 2020. URL: https://web.archive.org/web/20220520122512/https://file12.bitmain.com/shop-product-s3/firmware/4e25b493-58d5-4986-8cff-52006dda2038/2022/01/19/17/S19ServerManual.pdf.

23 Bitmain Technologies Holding Company. Difference between low power mode and low power enhanced mode, 2022. URL: https://web.archive.org/web/20220309092700/https://support.bitmain.com/hc/en-us/articles/360019738593-Difference-between-Low-Power-Mode-and-Low-Power-Enhanced-Mode.

24 Bitmain Technologies Holding Company. Recommended antminer monitor and management tools (apminertool & btc tool), 2022. URL: https://web.archive.org/web/20220309090337/https://support.bitmain.com/hc/en-us/articles/360023257293-Recommended-Antminer-monitor-and-management-tools-APMinerTool-BTC-Tool-.

25 BRAIINS Bitcoin Mining Company. Bitcoin mining profitability calculator, 2022. URL: https://web.archive.org/web/20220714050553/https://insights.braiins.com/en/profitability-calculator/.

26 BRAIINS Mining Company. Mining insights, 2022. URL: https://web.archive.org/web/20220818165736/https://insights.braiins.com/en/.

27 Tom Copeland and Vladimir Antikarov. *Real options*. Texere New York, New York, NY, USA, 2001.

28 John C Cox, Stephen A Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of financial Economics*, 7(3):229–263, 1979. doi:10.1016/0304-405x(79)90015-1.

29 CryptoCompare. Bitcoin mining profitability calculator, 2022. URL: https://web.archive.org/web/20220513095414/https://www.cryptocompare.com/mining/calculator/btc?HashingPower=40&HashingUnit=TH/s&PowerConsumption=1500&CostPerkWh=0.12&MiningPoolFee=1.

30 Quynh H Dang et al. Secure hash standard, 2015. doi:10.6028/nist.fips.180-4.

31 Nicola Dimitri. Bitcoin mining as a contest. *Ledger*, 2(0):31–37, 2017. doi:10.5195/ledger.2017.96.

32 Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, volume 61, pages 436–454. Springer, Association for Computing Machinery (ACM), June 2014. doi:10.1145/3212998.

33 Amanda Fabiano. Today @bitcoinbeezy gave a great talk at the @bitmaintech event focusing on how we, as a financing company, evaluate miners, 2022. URL: https://web.archive.org/web/20220726183620/https://twitter.com/_amanda_fab/status/1551999454433132544.

34 Amos Fiat, Anna Karlin, Elias Koutsoupias, and Christos Papadimitriou. Energy equilibria in proof-of-work mining. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, EC '19, pages 489–502, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3328526.3329630.

35 Cambridge Centre for Alternative Finance. Cambridge bitcoin electricity consumption index, 2022. URL: https://web.archive.org/web/20220818013329/https://ccaf.io/cbeci/index.

36 Yotam Gafni and Aviv Yaish. Greedy transaction fee mechanisms for (non-)myopic miners, 2022. doi:10.48550/ARXIV.2210.07793.

37 Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 281–310, Berlin, Heidelberg, 2015. Springer.

38 Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. Decentralization in bitcoin and ethereum networks. In Sarah Meiklejohn and Kazue Sako, editors, *Financial Cryptography and Data Security*, pages 439–457, Berlin, Heidelberg, 2018. Springer.

39 Guy Goren and Alexander Spiegelman. Mind the mining. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, EC '19, pages 475–487, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3328526.3329566`.

40 Samuel Haig. Btc hash rate slumps amid seasonal miner migration in china, 2020. URL: `https://cointelegraph.com/news/btc-hash-rate-slumps-amid-seasonal-miner-migration-in-china`.

41 Timo Hanke. AsicBoost - A Speedup for Bitcoin Mining, April 2016. `arXiv:1604.00575`.

42 Colin Harper and Ethan Vera. Hashrate index 2021 year-end report, 2022. URL: `https://web.archive.org/web/20220113191729/https://blog.hashrateindex.com/content/files/2022/01/Hashrate-Index-2021-Year-End-Report.pdf`.

43 Yoshinori Hashimoto and Shunya Noda. Pricing of mining asic and its implication to the high volatility of cryptocurrency prices, 2019. `doi:10.2139/ssrn.3368286`.

44 Adam Hayes. The decision to produce altcoins: Miners' arbitrage in cryptocurrency markets, December 2014. `doi:10.2139/ssrn.2579448`.

45 Adam S. Hayes. Cryptocurrency value formation: An empirical study leading to a cost of production model for valuing bitcoin. *Telematics and Informatics*, 34(7):1308–1321, 2017. `doi:10.1016/j.tele.2016.05.005`.

46 Christopher Helman. How this billionaire-backed crypto startup gets paid to not mine bitcoin, 2020. URL: `https://web.archive.org/web/20200524160256/https://www.forbes.com/sites/christopherhelman/2020/05/21/how-this-billionaire-backed-crypto-startup-gets-paid-to-not-mine-bitcoin/#7bdc51b97596`.

47 Hashrate Index. Bitcoin hashprice index, 2022. URL: `https://web.archive.org/web/20220714205330/https://data.hashrateindex.com/chart/bitcoin-hashprice-index`.

48 Hashrate Index. Profitability calculator, 2022. URL: `https://web.archive.org/web/20220623021224/https://hashrateindex.com/tools/calculator`.

49 Harold Jeffreys. *The theory of probability*. OUP Oxford, Oxford, UK, 1998. `doi:10.2307/2669965`.

50 Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, New York, December 2020. `doi:10.1201/9781351133036`.

51 Sishan Long, Soumya Basu, and Emin Gün Sirer. Measuring miner decentralization in proof-of-work blockchains, 2022. `doi:10.48550/ARXIV.2203.16058`.

52 Robert Merton. Theory of rational option pricing. *Bell Journal of Economics*, 4(1):141–183, 1973.

53 minerstat. Mining profitability calculator, 2022. URL: `https://web.archive.org/web/20220530152147/https://minerstat.com/mining-calculator`.

54 Compass Mining. What is hashprice?, 2021. URL: `https://web.archive.org/web/20211019011808/https://compassmining.io/education/what-is-hashprice/`.

55 Michael Mirkin, Yan Ji, Jonathan Pang, Ariah Klages-Mundt, Ittay Eyal, and Ari Juels. Bdos: Blockchain denial-of-service. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, pages 601–619, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3372297.3417247`.

56 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: `https://web.archive.org/web/20100704213649/https://bitcoin.org/bitcoin.pdf`.

57 John Naughton. As energy prices soar, the bitcoin miners may find they have struck fool's gold, 2022. URL: `https://web.archive.org/web/20220717043121/https://www.theguardian.com/commentisfree/2022/jun/11/as-energy-prices-soar-the-bitcoin-miners-may-find-they-have-struck-fools-gold`.

58 Bloomberg News. Bitcoin 'hash crash' rebound points to miners plugging back in, 2021. URL: https://www.bloomberg.com/news/articles/2021-09-03/bitcoin-hash-crash-rebound-points-to-miners-plugging-back-in.

59 NiceHash. Profitability calculator, 2022. URL: https://web.archive.org/web/20220519093350/https://www.nicehash.com/profitability-calculator.

60 Lumerin Protocol. What is hashprice?, 2022. URL: https://web.archive.org/web/20220314144723/https://medium.com/lumerin-blog/what-is-hashprice-9651cb08b215.

61 Michel Rauchs and Garrick Hileman. *Global Cryptocurrency Benchmarking Study*. Cambridge Centre for Alternative Finance, Cambridge Judge Business School, University of Cambridge, Cambridge, United Kingdom, 2017. URL: https://EconPapers.repec.org/RePEc:jbs:altfin:201704-gcbs.

62 Jamie Redman. Chinese bitcoin miners migrate north after wet season, 2019. URL: https://web.archive.org/web/20220427014036/https://news.bitcoin.com/chinese-bitcoin-miners-migrate-north-after-wet-season/.

63 M. Rosenfeld. Analysis of Bitcoin Pooled Mining Reward Systems, December 2011. arXiv:1112.4980.

64 Mark Rubinstein. Implied binomial trees. *The journal of finance*, 49(3):771–818, 1994. doi:10.1111/j.1540-6261.1994.tb00079.x.

65 M. Salimitari, M. Chatterjee, M. Yuksel, and E. Pasiliao. Profit maximization for bitcoin pool mining: A prospect theoretic approach. In *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*, pages 267–274, San Francisco, CA, USA, October 2017. IEEE. doi:10.1109/CIC.2017.00043.

66 Okke Schrijvers, Joseph Bonneau, Dan Boneh, and Tim Roughgarden. Incentive compatibility of bitcoin mining pool reward functions. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security*, pages 477–498, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

67 Spencer Sherwood. Why bitcoin asic prices can reach new highs in 2022, 2022. URL: https://web.archive.org/web/20220127160904/https://miningstore.com/understanding-the-bitcoin-mining-rig-market/why-bitcoin-asic-prices-can-reach-new-highs-in-2022/.

68 MacKenzie Sigalos. As major winter storm descends on texas, bitcoin miners are helping the power grid brace for impact, 2022. URL: https://web.archive.org/web/20220203143819/https://www.cnbc.com/2022/02/03/winter-storm-descends-on-texas-bitcoin-miners-shut-off-to-protect-ercot.html.

69 Paulo Silva, David Vavricka, João Barreto, and Miguel Matos. Impact of geo-distribution and mining pools on blockchains: A study of ethereum. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 245–252, San Francisco, CA, USA, June 2020. IEEE. doi:10.1109/DSN48063.2020.00041.

70 similarweb. Website traffic - check and analyze any website, 2022. URL: https://web.archive.org/web/20220818075305/https://www.similarweb.com/.

71 Rajani Singh, Ashutosh Dhar Dwivedi, Gautam Srivastava, Agnieszka Wiszniewska-Matyszkiel, and Xiaochun Cheng. A game theoretic analysis of resource mining in blockchain. *Cluster Computing*, 23(3):2035–2046, 2020. doi:10.1007/s10586-020-03046-w.

72 Yonatan Sompolinsky and Aviv Zohar. Bitcoin's security model revisited, 2016. arXiv:1605.09193.

73 Vikram B Suresh, Sudhir K Satpathy, and Sanu K Mathew. Optimized sha-256 datapath for energy-efficient high-performance bitcoin mining, November 27 2018. US Patent 10,142,098.

74 taserz. Asic.to firmware s17+ 95th/s • t17+ 80th/s t17 40w/t • s17/t17 on over 250k asic, 2019. URL: https://web.archive.org/web/20210812103613/https://bitcointalk.org/index.php?topic=5208500.0.

75 Layer1 Technologies. Building bitcoin batteries, 2019. URL: https://web.archive.org/web/20220401125045/https://layer1.com/.

76 Natkamon Tovanich, Nicolas Soulié, and Petra Isenberg. Visual analytics of bitcoin mining pool evolution: On the road toward stability? In *2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, San Francisco, CA, USA, April 2021. IEEE. doi:10.1109/NTMS49979.2021.9432675.

77 Lenos Trigeorgis et al. *Real options: Managerial flexibility and strategy in resource allocation.* MIT press, Cambridge, MA, USA, 1996.

78 Lenos Trigeorgis and Jeffrey J Reuer. Real options theory in strategic management. *Strategic Management Journal*, 38(1):42–63, 2017.

79 Itay Tsabary and Ittay Eyal. The gap game. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 713–728, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3243734.3243737.

80 ASIC Miner Value. Asic miner value, 2021. URL: https://www.asicminervalue.com/.

81 ASIC Miner Value. Miners profitability, 2022. URL: https://web.archive.org/web/20220808050607/https://www.asicminervalue.com/.

82 Ethan Vera. What is bitcoin mining firmware?, 2020. URL: https://web.archive.org/web/20220520114305/https://blog.hashrateindex.com/asic-custom-firmware-guide/.

83 VNISH. Firmware for overclocking and downvolt antminer s17pro s17 s17+, 2022. URL: https://web.archive.org/web/20220520134308/https://vnish-firmware.com/en/razgon-antminer-s17-s17pro-s17/.

84 Gian M. Volpicelli. As bitcoin falters, crypto miners brace for a crash, May 2022. URL: https://web.archive.org/web/20220531110502/https://www.wired.co.uk/article/bitcoin-mining-crisis.

85 Canhui Wang, Xiaowen Chu, and Yang Qin. Measurement and analysis of the bitcoin networks: A view from mining pools. In *2020 6th International Conference on Big Data Computing and Communications (BIGCOM)*, pages 180–188, San Francisco, CA, USA, 2020. IEEE, IEEE. doi:10.1109/bigcom51056.2020.00032.

86 whattomine. Crypto coins mining profit calculator, 2022. URL: https://web.archive.org/web/20220729175135/https://whattomine.com/.

87 Bitcoin Wiki. Comparison of mining pools, 2022. URL: https://web.archive.org/web/20220819173306/https://en.bitcoin.it/wiki/Comparison_of_mining_pools.

88 Aviv Yaish, Gilad Stern, and Aviv Zohar. Uncle maker: (time)stamping out the competition in ethereum. In *Proceedings of the 2023 ACM SIGSAC Conference on Computerand Communications Security (CCS '23)*, CCS '23, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3576915.3616674.

89 Aviv Yaish, Saar Tochner, and Aviv Zohar. Blockchain stretching & squeezing: Manipulating time for your best interest. In *Proceedings of the 23rd ACM Conference on Economics and Computation*, EC '22, pages 65–88, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3490486.3538250.

90 Aviv Yaish and Aviv Zohar. Pricing asics for cryptocurrency mining, 2023. arXiv:2002.11064.

91 Kristina Zucchi. Is bitcoin mining profitable?, July 2022. URL: https://web.archive.org/web/20220815040240/https://www.investopedia.com/articles/forex/051115/bitcoin-mining-still-profitable.asp.

F3B: A Low-Overhead Blockchain Architecture with Per-Transaction Front-Running Protection

Haoqian Zhang
École Polytechnique Fédérale de Lausanne, Switzerland

Louis-Henri Merino
École Polytechnique Fédérale de Lausanne, Switzerland

Ziyan Qu
École Polytechnique Fédérale de Lausanne, Switzerland

Mahsa Bastankhah
École Polytechnique Fédérale de Lausanne, Switzerland

Vero Estrada-Galiñanes
École Polytechnique Fédérale de Lausanne, Switzerland

Bryan Ford
École Polytechnique Fédérale de Lausanne, Switzerland

Abstract

Front-running attacks, which benefit from advanced knowledge of pending transactions, have proliferated in the blockchain space since the emergence of decentralized finance. Front-running causes devastating losses to honest participants and continues to endanger the fairness of the ecosystem. We present Flash Freezing Flash Boys (F3B), a blockchain architecture that addresses front-running attacks by using threshold cryptography. In F3B, a user generates a symmetric key to encrypt their transaction, and once the underlying consensus layer has finalized the transaction, a decentralized secret-management committee reveals this key. F3B mitigates front-running attacks because, before the consensus group finalizes it, an adversary can no longer read the content of a transaction, thus preventing the adversary from benefiting from advanced knowledge of pending transactions. Unlike other mitigation systems, F3B properly ensures that all unfinalized transactions, even with significant delays, remain private by adopting per-transaction protection. Furthermore, F3B addresses front-running at the execution layer; thus, our solution is agnostic to the underlying consensus algorithm and compatible with existing smart contracts. We evaluated F3B on Ethereum with a modified execution layer and found only a negligible (0.026%) increase in transaction latency, specifically due to running threshold decryption with a 128-member secret-management committee after a transaction is finalized; this indicates that F3B is both practical and low-cost.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Blockchain, DeFi, Front-running Mitigation

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.3

Related Version *Full Version*: `https://arxiv.org/abs/2205.08529` [65]

Funding This research was supported in part by U.S. Office of Naval Research grant N00014-19-1-2361, the AXA Research Fund, the PAIDIT project funded by ICRC, the IC3-Ethereum Fund, Algorand Centres of Excellence programme managed by Algorand Foundation, and armasuisse Science and Technology. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding sources.

Acknowledgements The authors wish to thank Cristina Basescu, Pasindu Nivanthaka Tennage, Pierluca Borsò-Tan, and Simone Colombo for their extremely helpful comments and suggestions and especially thank Shufan Wang for prototyping F3B on the Ethereum blockchain.

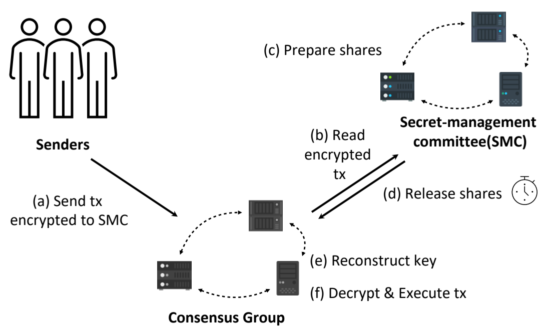


Figure 1 F3B architecture. Senders publish encrypted transactions to the consensus group. The secret-management committee releases the decryption shares once the transactions are no longer pending. Finally, the consensus group reconstruct the key and decrypt and execute the transaction. The secret-management committee and the consensus group can consist of the same set of servers. For clarity in this paper, we logically separate them into two different entities.

1 Introduction

Front-running is the practice of benefiting from the advanced knowledge of pending transactions [20, 7, 1]. Although benefiting some entities involved, this practice puts others at a significant financial disadvantage, making this behavior illegal in traditional markets with established securities regulations [20].

However, the open and pseudonymous nature of blockchain transactions and the difficulties of pursuing miscreants across numerous jurisdictions have made front-running attractive, particularly in decentralized finance (DeFi) [38, 20, 15]. Front-running actors in the blockchain space can read the contents of pending transactions and benefit from them by, *e.g.*, creating their own transactions and positioning them according to the target transaction [4, 15, 20].

Front-running negatively impacts honest DeFi actors and endangers the fairness of this multi-billion market [18]. One estimate suggests that front-running attacks amount to $280 million in losses for DeFi actors each month [42]. Front-running also threatens the underlying consensus layer's security by incentivizing unnecessary forks [17, 15].

Despite work addressing front-running, several unmet challenges exist, such as high latency, being restricted to a specific environment, or raising security concerns. Namecoin, an early example of mitigating front-running attacks by having users send a commit and later a reveal transaction, requires two rounds of communication with the underlying blockchain [29]. Submarine further improves Namecoin's design by hiding the addresses of smart contracts involved, but it induces three rounds of communication to the underlying blockchain [39, 29]. Both approaches induce high latency. Other works have taken a different approach to mitigate front-running attacks by tailoring their solution to a specific application or consensus algorithm [13, 52, 60, 5, 2, 3, 40, 25].

A promising approach is to use threshold encryption, where clients encrypt their transactions to prevent malicious actors from understanding those transactions, as presented in Fairblock [43] and Shutter [57, 58]. However, these schemes require clients to choose a future block to derive the encryption key, which raises security concerns. Suppose a transaction failed to be finalized in the client-chosen block due to, for example, a crypto mania that overwhelms the blockchain network [31] or a deliberate denial of service attack [20]. In this case, the transaction is undesirably revealed (see Section 3.3 for details).

We present Flash Freezing Flash Boys[1] (F3B), a novel blockchain architecture with front-running protection that has a low latency overhead and is compatible with existing consensus algorithms and smart contract implementations. Like Fairblock [43] and Shutter [57, 58], F3B addresses front-running by adopting threshold encryption, but it accomplishes this on a **per-transaction** basis rather than a per-block basis. Rather than selecting an encryption key linked to a future block, clients generate an encryption key for each transaction. This ensures that a transaction remains confidential until the block containing the transaction has received enough confirmations.

As described in Figure 1, F3B's architecture consists of the following steps: (a) A client encrypts their transaction to a secret-management committee (SMC) and sends their encrypted transaction to the consensus group that operates the underlying blockchain. (b) The SMC reads the encrypted transaction from the underlying blockchain. (c) The SMC prepares the decryption shares for the consensus group. (d) The SMC releases the decryption shares to the consensus group once the underlying blockchain has finalized the transaction. (e) The consensus group reconstructs the key. (f) The consensus group decrypts and executes the transaction. Once the SMC begins to release the decryption shares, malicious actors cannot launch a front-running attack because the transaction is already irreversibly ordered on the blockchain. Although adversaries may attempt to run *speculative* front-running attacks, where they guess the contents of a transaction on metadata information like the sender's address, these attacks are more likely to fail and can prove to be unprofitable [4]. Nonetheless, we discuss mitigation solutions for these attacks in Section 10.4.

F3B addresses two key practical challenges: (a) mitigating spamming of inexecutable encrypted transactions onto the underlying blockchain, and (b) limiting latency overhead. To mitigate spamming, we introduce a deposit-refund storage fee for storing encrypted transactions, along with the standard execution fee (*e.g.*, gas in Ethereum). To limit the latency overhead, users write only data onto the underlying blockchain once to achieve front-running protection.

We propose two cryptographic threshold schemes that can plug into F3B: TDH2[56] and PVSS [53]. TDH2 enables clients to encrypt their transactions under the same public key of a secret-management committee which is only changeable by time-consuming DKG or resharing protocols. On the other hand, PVSS empowers clients to adopt a different secret-management committee for each transaction but at the cost of the additional preprocessing time for preparing the shares for each transaction.

We implemented a prototype of F3B with post-Merge[2] Ethereum [23] as the underlying blockchain and Dela [19] as the secret-management committee. We measure the latency overhead by comparing the time it takes to decrypt and execute a transaction with the time it takes just to execute the transaction. Our analysis shows that, with a committee size of 128, the latency overhead is 0.026% and 0.027% for Ethereum under the TDH2 and PVSS respectively; In comparison, Submarine, which also offers per-transaction protection and hides the address of smart contracts as F3B, exhibits a 200% latency overhead, as it requires three rounds of communication with the underlying blockchain [39, 11]. For part of our prototype, we modified Ethereum's execution layer by adding a new transaction type featuring encryption and delayed execution. By only modifying the execution layer, we

[1] The name *Flash Boys* comes from a popular book revealing this aggressive market-exploiting strategy on Wall Street in 2014 [38].

[2] The Merge refers to the merge executed on September 15th, 2022, to complete Ethereum's transition to proof-of-stake consensus.

can (a) provide compatibility with various consensus algorithms embedded in Ethereum's consensus layer, including Proof-of-Work (PoW), Proof-of-Authority (PoA) and the recently added, Proof-of-Stake (PoS) and (b) protect existing smart contracts without requiring any code modifications.

In this paper, our key contributions are as follows:

1. The design of a blockchain architecture with front-running protection that uses threshold encryption on a per-transaction basis, enabling confidentiality for all pending transactions, even if transactions are delayed, while achieving low overhead.
2. The design of two protocols based on TDH2 and PVSS for F3B satisfies various demands and user scenarios.
3. A prototype that, on Ethereum's execution layer, demonstrates F3B's ability to be agnostic to (a) the underlying consensus algorithm and (b) to smart contract implementations while achieving low-latency overhead.
4. A systematic evaluation of F3B on post-Merge Ethereum by looking at transaction latency, throughput, and reconfiguration costs.

2 Background

In this section, we present a brief background on blockchain and smart contracts, and we introduce front-running attacks and mitigation strategies.

2.1 Blockchain & Transaction Ordering

A blockchain is an immutable append-only ledger of ordered transactions [44]. However, transactions go through a series of stages before they are finalized – irreversibly ordered – on the blockchain. After a sender creates a transaction, they need to propagate the transaction among the consensus nodes that then place the transaction in a pool of pending transactions, most commonly known as mempool. Notably, these transactions are *not yet irreversibly ordered*, thus opening up the possibility for front-running attacks. Furthermore, under certain probabilistic consensus algorithms, such as PoW or PoS, a transaction inserted onto the blockchain can still be reordered by inducing a fork of the underlying blockchain. Hence, to guarantee irreversible ordering for probabilistic consensus algorithms, a transaction must receive enough block confirmations – the number of blocks succeeding the block containing the transaction [44, 34, 14].

2.2 Smart Contract & Decentralized Exchange

A smart contract is an executable computer program modeled after a contract or an agreement that executes automatically [50]. A natural fit for smart contracts is on top of decentralized fault-tolerant consensus algorithms, such as PBFT-style algorithms, PoW, and PoS, to ensure their execution and integrity [63, 44, 32].

Although Bitcoin uses a form of smart contracts [44], it was not until Ethereum's introduction that the blockchain space realized Turing-complete smart contracts, the backbone necessary for creating complex decentralized exchanges. To interact with these complex smart contracts, users need to pay *gas*, a pseudo-currency that represents the execution cost by miners [21]. However, the expressiveness of smart contracts comes with significant risks, from inadvertent vulnerabilities to front-running. Front-running is exhibited by the lack of guarantees that the underlying blockchain provides regarding ordering.

2.3 Front-Running Attacks & Mitigation

The practice of front-running involves benefiting from advanced knowledge of pending transactions [20, 7, 1]. In itself, knowledge of pending transactions is harmless, but the ability to act on this information is where the true problem lies. In the context of blockchains, an adversary performs a front-running attack by influencing the order of transactions, provided that transactions in the mempool are entirely in the clear.

Cryptocurrencies suffer from mainly three types of front-running attacks [20]: displacement, insertion, and suppression. *Displacement* is the replacement of a target transaction with a new transaction formulated by the front-running attacker. *Insertion* is the malicious introduction of a new transaction before a target transaction in the finalized transaction ordering. *Suppression* is the long-term or indefinite delaying of a target transaction.

In an ideal world, front-running protection would consist of an *immediate* global ordering of each transaction, as clients broadcast their transactions to prevent attackers from changing their order. In reality, even if all participants were honest, such global ordering is practically impossible due to clock synchronization [16] and consistency problems (*e.g.*, two transactions having the same time). Malicious participants can still carry out front-running attacks, because timings can easily be manipulated.

A more practical solution involves encrypting transactions, thereby preventing the consensus group from knowing the contents of the transactions when ordering them. This solution mitigates front-running attacks as an attacker is hindered from taking advantage of pending *encrypted* transactions.

3 Strawman Protocols

In order to explore the challenges inherent in building a framework, such as F3B, we first examine a couple of promising but inadequate strawman approaches, representative of state-of-the-art proposals [39, 11, 43, 57] but simplified for expository purposes.

3.1 Strawman I: Sender Commit-and-Reveal

The first strawman design has the sender create two transactions: a *commit* and a *reveal* transaction. The commit transaction is simply a commitment (*e.g.*, hash) of the intended reveal transaction, which is simply the typical contents of a transaction that is normally vulnerable to front-running. The sender will propagate the commit transaction and then *wait* until its finality by the consensus group, before releasing the reveal transaction. Once the reveal transaction is propagated, the consensus group proceeds to verify and to execute the transaction, in the execution order that the commit transaction was finalized on the blockchain. Given the finality in the former transaction, the sender is unable to change the contents of the reveal transaction.

This simple strawman protocol mitigates front-running attacks because the commit transaction determines the execution order and the contents of the commit transaction do not expose the contents of the reveal transaction. However, this strawman protocol presents some notable challenges: (a) the sender must remain online to continuously monitor the blockchain to know when to release their reveal transaction, (b) the reveal transaction might be delayed due to a congestion event like the cryptokitties mania [31] or a deliberate denial-of-service (DoS) attack like the Fomo3D incident [20], (c) this approach is subject to output bias, as the consensus nodes or the sender can deliberately choose not to reveal certain transactions during the reveal phase [4], such as only revealing profitable ones and aborting others, and (d) this approach has a significant latency overhead of over 100%, given that the sender must now send two non-overlapping transactions instead of the one standard transaction.

3.2 Strawman II: The Trusted Custodian

A straightforward method for removing the sender from the equation, after sending the commit transaction, is to employ a trusted custodian. After the consensus group finalizes the transaction onto the underlying blockchain, the trusted custodian reveals the transaction's contents.

This strawman protocol mitigates front-running attacks, as the nodes cannot read, before ordering, the contents of the transaction. However, the trusted custodian presents a single point of failure: Consensus nodes cannot decrypt and execute a transaction if the custodian crashes. Instead, by employing a *decentralized* custodian, we can mitigate the single point of failure issues.

3.3 Strawman III: Threshold Encryption with Block Key

The next natural step is to have a decentralized committee that generates a public key for each block, thus enabling a user to encrypt their transaction for a future block. The committee would then release the private key after the block finality. Furthermore, the committee can use identity-based encryption [55] to enable users to derive a future block key based on the block's height.

This strawman protocol seems to mitigate front-running, as the transactions in a block are encrypted until they are finalized in their intended block. However, if an encrypted transaction fails to be included in the specified block, its contents will be revealed shortly thereafter while remaining unfinalized, thus making it vulnerable to front-running. Blockchain networks have repeatedly observed such failures due to congestion, such as cryptokitties manias [31], or well-funded DoS attacks, such as the Fomo3D attack that flooded the Ethereum network with transactions for three minutes [20]. Such an approach can incentivize a consensus node to intentionally produce an empty block by aiming to reveal the pending transactions for that block. Therefore, we require a per-transaction rather than a per-block level of confidentiality, thus ensuring that a transaction is never revealed before it is finalized on the blockchain.

4 System Overview

In this section, we present F3B's system goals, architecture, and models.

4.1 System Goals

Our system goals, inspired by our strawman protocols, are

- **Front-Running Protection:** prevents entities from practicing front-running.
- **Decentralization:** mitigates a single point of failure or compromise.
- **Confidentiality:** reveals a transaction, only after the underlying consensus layer finalizes it.
- **Compatibility:** remains agnostic to the underlying consensus algorithm and to smart contract implementation.
- **Low-Latency**: exhibits low-latency transaction-processing overhead.

4.2 Architecture Overview

F3B, shown in Figure 1, mitigates front-running attacks by working with a secret-management committee to manage the storage and release of on-chain secrets. Instead of propagating their transactions in cleartext, the sender can now encrypt their transactions and store the corresponding secret keys with the secret-management committee. Once the transaction is finalized, the secret-management committee releases the secret keys so that consensus nodes of the underlying blockchain can verify and execute transactions. Overall, the state machine replication of the underlying blockchain is achieved in two steps: the first is about the ordering of transactions, and the second is about the execution of transactions. As long as most trustees in the secret-management committee are secure and honest and the key is revealed to the public when appropriate, each consensus node can always maintain the same blockchain state.

F3B encrypts the entire transaction[3], such as the smart contract address, inputs, sender's signature, and other metadata, as those information can provide enough information to launch a probabilistic front-running attack, such as the Fomo3D attack [20] or a speculative attack based on the leakage of metadata [4].

4.3 System and Network Model

F3B's architecture consists of three components: *senders* that publish (encrypted) transactions, the *secret-management committee* (SMC) that manages and releases secrets, and the *consensus group* that maintains the underlying blockchain. For the F3B based on the PVSS scheme, the client can choose a different SMC for each transaction. For the F3B based on the THD2 scheme, an SMC has a fixed membership over one epoch. When transiting from one epoch to the next, the SMC can modify its membership under the THD2 scheme with backward secrecy to prevent new trustees from decrypting old transactions without interrupting users' encryption by running a resharing protocol [62].

The secret-management committee and the consensus group can consist of the same set of servers. For clarity in this paper, we logically separate them into two different entities.

For the underlying network, we assume that all honest blockchain nodes and trustees of the SMC are well connected and that their communication channels are synchronous, *i.e.*, if an honest node or trustee broadcasts a message, then all honest nodes and trustees receive the message within a known maximum delay [46].

4.4 Threat Model

We assume that the adversary is computationally bounded, that the cryptographic primitives we use are secure, and in particular that the Diffie-Hellman problem and its decisional variant are hard. We further assume that all messages are digitally signed and that the consensus nodes and the SMC only process correctly signed messages.

The secret management committee consists of n trustees, where f can fail or behave maliciously. We require $n \geq 2f+1$ and set the secret-recovery threshold to $t = f+1$. We assume that the underlying blockchain is secure: *e.g.*, at most f' of $3f'+1$ validators can fail or misbehave in a PBFT-style or PoS blockchain, or the adversary controls less than 50% computational power in a PoW blockchain. We acknowledge that the security assumptions for the secret management committee and the underlying blockchain might differ, potentially reducing the overall system's security to the least secure subsystem.

We assume that attackers do not launch speculative front-running attacks [4], but we present a discussion on some mitigation strategies for reducing side-channel leakage in Section 10.4.

[3] Section 10.4 further discusses how to hide the sender's address.

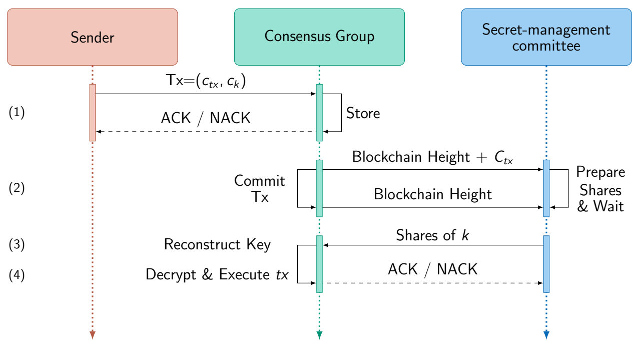


Figure 2 F3B per-transaction protocol steps: (1) Send an encrypted transaction to the underlying blockchain, (2) Prepare shares by trustees while waiting for transaction finality, (3) Reconstruct the key, (4) Execute the transaction.

5 F3B Protocol

In this section, we introduce the F3B's protocol, starting with some preliminaries, followed by the F3B's detailed design. The full paper [65] offers a more comprehensive protocol description, and Section 10 introduces some optimizations.

5.1 Preliminaries

In this subsection, we introduce our preliminaries, including our baseline model for the underlying blockchain and the cryptographic primitives used in F3B.

Blockchain Model

To compare F3B's impact, we model the underlying blockchain to involve a consensus protocol that finalizes transactions into a block that is linked to a previous block. We assume the underlying block's time as L_b seconds. In PoW and PoS-based blockchains, a transaction is finalized only after a certain number of additional blocks have been added to the chain (also known as block confirmations). Thus, we define that a transaction is finalized after m block confirmations. Therefore, our baseline transaction latency is mL_b.

Shamir's Secret Sharing

A (t, n)-threshold secret sharing scheme enables a dealer to share a secret s among n trustees such that any group of $t \leq n$ or more trustees can reconstruct s and no group less than t trustees learns any information about s. Whereas a simple secret sharing scheme assumes an honest dealer, verifiable secret sharing (VSS) enables the trustees to verify that the shares they receive are valid [24]. Public verifiable secret sharing (PVSS) further improves VSS to enable a third party to check all shares [53].

Distributed Key Generation (DKG)

DKG is a multi-party (t, n) key-generation process for collectively generating a private-public key pair (sk, pk), without relying on a single trusted dealer; each trustee i obtains a share sk_i of the secret key sk, and collectively obtains a public key pk [27]. Any client can now use pk to encrypt a secret, and at least t trustees must cooperate to retrieve this secret [56].

5.2 Protocol Outline

We present the outline of F3B protocols with two different threshold cryptographic schemes. Figure 2 presents the protocol outline, and the full paper [65] offers a more comprehensive protocol description.

5.2.1 Protocol based on TDH2

Setup

Before an epoch, the secret-management committee runs a DKG protocol to generate a private key share sk^i_{smc} for each trustee and a collective public key pk_{smc} written onto the underlying blockchain. To offer chosen-ciphertext attack protection and to verify the correctness of secret shares, we utilize the TDH2 cryptosystem [56] containing NIZK proofs.

Per-Transaction Protocol

1. *Write Transaction:* A sender first generates a symmetric key k and encrypts it with pk_{smc} from the underlying blockchain, thus obtaining the resulting ciphertext c_k. Next, the sender creates their signed transaction and symmetrically encrypts it by using k, denoted as $c_{tx} = enc_k(tx)$. Finally, the sender sends (c_{tx}, c_k) to the consensus group who writes the pair onto the blockchain.
2. *Shares Preparation by Trustees:* Once written, each secret-management committee trustee reads c_k from the sender's transaction and prepares their decrypted share of k.
3. *Key Reconstruction:* When the sender's transaction (c_{tx}, c_k) is finalized onto the underlying blockchain (after m block confirmations), each secret-management committee trustee releases their share to the consensus group. The consensus group verifies the decrypted shares and uses them to reconstruct k by Lagrange interpolation of shares when there are at least t valid shares.
4. *Decryption and Execution:* The consensus group finally symmetrically decrypts the transaction $tx = dec_k(c_{tx})$ using k, thus enabling it to execute tx.

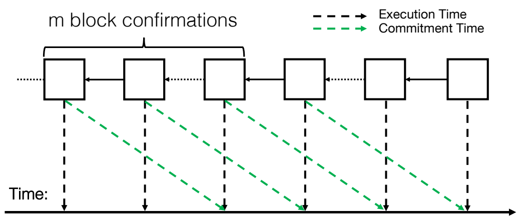
Resharing Protocol

To modify a SMC's membership and to offer backward secrecy over epochs, an SMC can periodically run a verifiable resharing protocol [62] to replace certain trustees or redistribute the trustees' private keys. Unlike DKG, resharing keeps the epoch's public key, thus preventing undesirable interruptions of encryption services.

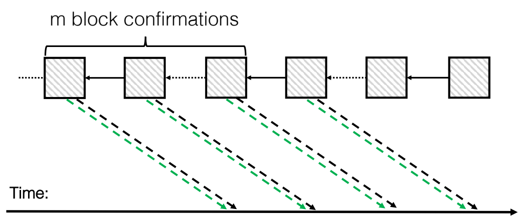
5.2.2 Protocol based on PVSS

Per-Transaction Protocol

0. *Share Preparation By sender:* For every transaction, the sender runs the PVSS protocol [53] to generate an encrypted key share $share_i$ for each trustee, as well as a corresponding NIZK proof and public polynomial commitment. The proof and commitment can be used to verify the correctness of key share and protect against chosen-ciphertext attacks. The sender obtains the symmetric key k from the PVSS protocol.



(a) Execution and finality time in Ethereum.



(b) Execution and finality time in F3B.

Figure 3 In Ethereum, once they are inserted in the blockchain, the transactions are executed and finalized after receiving m block confirmations. Whereas, in F3B, transactions are encrypted, and their executions are postponed after receiving m block confirmations when the secret-management committee releases the encryption keys. Both scenarios have a similar finality time.

1. *Write Transaction:* A sender first creates the ciphertext c_k with the key shares, NIZK proofs, and commitments generated during share preparation. Next, the sender creates their transaction and symmetrically encrypts it by using the symmetric key k, denoted as $c_{tx} = enc_k(tx)$. Finally, the sender sends (c_{tx}, c_k) to the consensus group who writes the pair onto the blockchain.
2. *Shares Preparation by Trustees:* Same as (2) in 5.2.1.
3. *Key Reconstruction:* Same as (3) in 5.2.1.
4. *Decryption and Execution:* Same as (4) in 5.2.1.

5.3 Overhead Analysis

We analyze both protocols' overheads. Write Transaction (step 1) is identical to sending a transaction to the underlying blockchain. We assume trustees can finish Shares Preparation by Trustees (step 2) within the confirmation time of the tx[4]. Hence, the time for steps 1 and 2 is equivalent to finalizing a transaction on the underlying blockchain and waiting until its finality, which takes mL_b time based on our baseline model (Section 5.1). As in PVSS protocol, the sender can finish Share Preparation By sender (step 0) before having the tx; thus step 0 does not contribute to the transaction latency. Comparing our protocol with the baseline, Key Reconstruction (step 3) and Decryption and Execution (step 4) are additional steps, and we denote the time of those steps to be L_r.

Figure 3 demonstrates the conceptual difference in finality and execution time between F3B and the baseline. As the secret-management committee releases the secret keys with a delay of m blocks, this introduces an execution delay of m blocks. However, in both cases, to prevent attacks such as double-spending, the recipient should not accept a transaction, until it is finalized. Therefore on a commercial level, F3B is similar to the baseline because it exhibits the finality time of a transaction that is of use to the recipient[5].

[4] As we presented in Section 9, confirmation time in Ethereum is much longer than the share preparation time by trustees.

[5] In F3B, transaction finalization is slower due to the key reconstruction and delayed execution after transaction finality. However, the overhead is negligible compared to finality time, as discussed in Section 9.

5.4 TDH2 and PVSS: Pros and Cons

When applying THD2 and PVSS to F3B, each scheme has some advantages and disadvantages. This subsection offers qualitative comparisons, whereas Section 9 provides quantitative comparisons between the two protocols.

- **Preprocessing:** In TDH2, the secret-management committee needs to do DKG per epoch, whereas in PVSS, the sender needs to prepare shares per transaction.
- **Membership:** In TDH2, the secret-management committee's membership is fixed per epoch, whereas in PVSS, the sender can choose a different secret-management committee for each transaction, providing the best flexibility.
- **Ciphertext:** TDH2 has a constant ciphertext length, whereas PVSS's ciphertext grows linearly with the size of the secret-management committee.

In conclusion, no one protocol can completely replace another. System designers need to choose one or both protocols based on their needs and constraints to mitigate front-running attacks.

6 Achieving the System Goals

In this section, we present how F3B achieves the system goals outlined in Section 4.1.

Front-Running Protection: ***prevents entities from practicing front-running.*** We reason the protection offered by F3B from the definition of front-running: if an adversary cannot benefit from pending transactions, he cannot launch front-running attacks. In F3B, the sole entity that knows the content of a pending transaction is the sender who is financially incentivized to *not* release its contents. The content is revealed only when its transaction is finalized; thus, by definition, the attacker has no means to launch a front-running attack. However, we acknowledge that attackers can use side channels (*e.g.* metadata such as sender's address and transaction size) of the encrypted transaction to launch *speculative* front-running attacks, as discussed in Section 4.4 and Section 10.4. We present a more comprehensive security analysis discussion in Section 7.

Decentralization: ***mitigates a single point of failure or compromise.*** Due to the properties of DKG [27], THD2 [56], and PVSS [53], the SMC can handle up to $t-1$ malicious trustees and up to $n-t$ offline trustees.

Confidentiality: ***reveals a transaction, only after the underlying consensus layer finalizes it.*** The sender encrypts each transaction with a newly generated symmetric key. The symmetric key is (a) encrypted under the secret-management committee's public key in TDH2-based protocol, (b) embedded into the encrypted shares in PVSS-based protocol. In both protocols, $f+1$ trustees are required to retrieve the symmetric key. Per our threat model, only f trustees can behave maliciously; this ensures that the symmetric key cannot be revealed. We outline a more detailed security analysis in Section 7.

Compatibility: ***remains agnostic to the underlying consensus algorithm and to smart contract implementation.*** F3B requires modifying the execution layer to enable encrypted transactions. However, the consensus layer remains untouched, thus agnostic to the underlying consensus algorithms. Furthermore, F3B does not require to modify smart contract implementations, thus enabling existing smart contracts to benefit from front-running protection automatically.

Low-Latency: *exhibits low-latency transaction-processing overhead.* Similar to the baseline model, F3B requires clients to write only one transaction onto the underlying blockchain. This enables F3B to have a low-latency overhead compared to other front-running protection design that require multiple transactions for the same security guarantees. We present an evaluation of this latency overhead in Section 9.

7 Security Analysis

In this section, we introduce the security analysis of F3B's protocol.

7.1 Front-Running Protection

From our threat model, we reason about why an attacker can no longer launch front-running attacks with absolute certainty of a financial reward, even with the collaboration of at most f malicious trustees. As we assume that the attacker does not launch speculative attacks based on metadata of the encrypted transactions, the only way the attacker can front-run transactions is by using the plaintext content of the transaction. As the attacker cannot access the content of the transaction before it is finalized on the underlying blockchain, then the attacker cannot benefit from the pending transaction. This prevents front-running attacks (by the definition of front-running). As we assume that the symmetric encryption we use is secure, the attacker cannot decrypt the transaction based on its ciphertext. Due to the properties of TDH2 [56], DKG [27], and PVSS [53] with our threat model, the attacker cannot obtain the private key and/or reconstruct the symmetric key. Recall that the attacker can collude with at most only f trustees, and that $f+1$ are required to recover or gain information about the symmetric key.

7.2 Replay Attack

We consider a scenario in which an adversary can copy a pending (encrypted) transaction and submit it as their own transaction to reveal the transaction's contents, before the victim's transaction is finalized. By revealing the contents of the copied transaction, the attacker can then trivially launch a front-running attack. However, we explain the reason the adversary is unable to benefit from such a strategy.

In the first scenario, the adversary copies the ciphertext c_k and the encrypted transaction c_{tx} from $\mathrm{tx_w}$, then creates a new write transaction $\mathrm{tx}'_{\mathrm{w}}$, digitally signed with their signature. However, even if the adversary's $\mathrm{tx}'_{\mathrm{w}}$ is decrypted and executed before the victim's transaction $\mathrm{tx_w}$, it effectively results in the blockchain executing $\mathrm{tx_w}$[6]. This leaves the adversary with no time to front-run their own $\mathrm{tx}'_{\mathrm{w}}$ without knowing its contents.

In our second scenario, the adversary instead sends the transaction to a blockchain with smaller m block confirmations. Consider two blockchains b_1 and b_2 whose required number of confirmation blocks are m_1 and m_2 with $m_1 > m_2$. If the adversary changes the label L to L' for the blockchain b_2 instead of blockchain b_1, the secret-management committee will successfully decrypt the transaction. However, we argue it is hard to form a valid write-transaction with L' by the adversary.

[6] Note that tx is already a signed transaction; thus, $\mathrm{tx_w}$ and $\mathrm{tx}'_{\mathrm{w}}$ have the same effect on the blockchain.

For the TDH2 protocol, the adversary would need to generate $e' = \mathrm{H}_1\left(c, u, \bar{u}, w, \bar{w}, L'\right)$ and $f = s + re'$, without knowing the random parameter r and s. Suppose the adversary generates $u = g^r, \bar{u} = \bar{g}^{r'}$ with $r \neq r'$ and $w = g^s, \bar{w} = \bar{g}^{s'}$ with $s \neq s'$. For $\mathrm{tx}'_{\mathrm{w}}$ to be valid, we must have $g^f = wu^e$ and $\bar{g}^f = \bar{w}\bar{u}^e$, this implies that $(s - s') + e(r - r') = 0$. As $r \neq r'$, the adversary has only a negligible chance of having $\mathrm{tx}'_{\mathrm{w}}$ pass verification.

For the PVSS protocol, the adversary must replace the original generator h with h' derived from $H(L')$. Hence, the adversary has to do the proofs without secrets. The security of PVSS guarantees that they only have a negligible probability of succeeding. Note that the base point has to be random to ensure the security. Using Elligator maps [8] guarantees that the generator h is random.

8 Incentive

F3B must incentivize actors to operate and follow the protocol honestly. In this section, we address the critical incentives that, in F3B, prevent spamming transactions and that deter collaboration among trustees from prematurely revealing transactions.

8.1 Spamming Protection

As the consensus group cannot execute encrypted transactions, an adversary could, at a low cost, spam the blockchain with non-executable transactions (*e.g.*, inadequate fees, malformed transactions), thus delaying the finality of honest transactions. To make such an attack costly, we introduce a storage deposit, alongside the traditional execution fee (*e.g.*, gas in Ethereum) and adjustable based on the transaction's size. The underlying blockchain can deduct the storage deposit from the sender's balance, much like paying a transaction fee. Then the blockchain can partially refund the deposit after successful execution by the consensus nodes. This approach imposes a low-cost fee on compliant users and a penalty on those who misbehave.

8.2 Operational Incentive

We need a similar incentive structure for the secret-management committees; similar to the way consensus nodes are rewarded for following the blockchain protocol via an execution fee. Whereas SMCs could be rewarded using the execution fee, this fee does not prevent SMC trustees from colluding for their financial gain. For example, an SMC might silently collude with a consensus group by prematurely giving them the decryption shares. Given the difficulty of detecting out-of-band collusion, we need to discourage it from doing so by significantly rewarding anyone who can prove the existence of such collusion.

We propose an incentive structure, where we require each trustee in a secret-management committee to lock an amount c for collateral and, in exchange, they are rewarded proportionally to the staked amount ac for the services they provide. Remind that, based on our threat model (Section 4.4), t trustees must collude altogether to reconstruct a transaction. To maintain security, the potential gain that t trustees benefit from front-running must be less than the potential loss $(1 + a)ct$, which malicious trustees would incur through the slashing protocol described in Section 8.3. Hence, a higher potential loss value $(1 + a)ct$ ensures security for a longer epoch length. If other factors stay consistent, designers can support a longer epoch length by either increasing the collateral requirement (by raising c) or by involving more trustees (by increasing t).

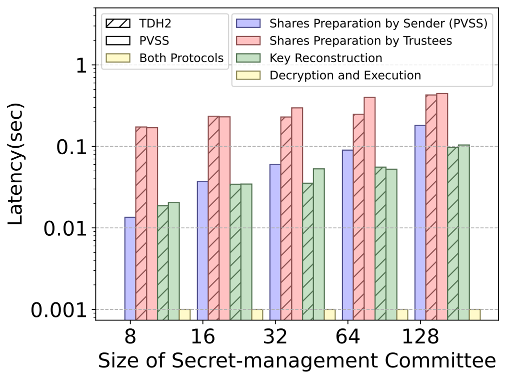


Figure 4 The breakdown latency of each step in F3B by varying the number of secret-management committee trustees from 8 to 128 nodes.

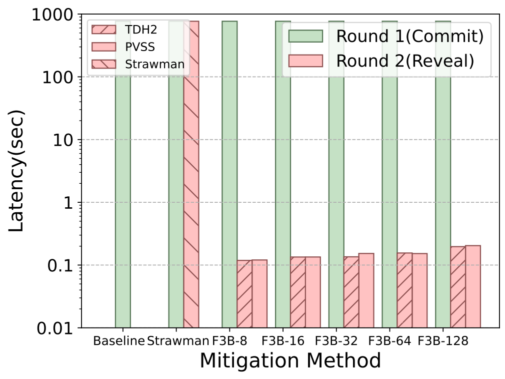


Figure 5 A comparison of the sender commit-and-reveal approach latency with F3B against a baseline modeled in Ethereum. The string "F3B-X" represents X trustees.

8.3 Slashing Protocol

We need to have a protocol that rewards anyone who can prove a trustee's or the entire secret-management committee's misbehavior to discourage the release of the shares prematurely. At the same time, we do not want anyone to accuse a secret-management committee or a specific trustee without repercussions if the SMC or trustee did not actually misbehave.

To accomplish our objective, each trustee of the secret-management committee must stake some amount of cryptocurrency in a smart contract that handles disputes between a defendant (the entire secret-management committee or a particular trustee) and a plaintiff. To start a dispute, the plaintiff will invoke the smart contract with the correct decryption share for a currently pending transaction and their own stake. Suppose the smart contract validates that this is a correct decryption share and that the dispute started before the transaction in question was revealed by the secret-management committee. In this case, the defendant's stake is forfeited and sent to the plaintiff.

At a protocol level, to prove a correct decryption share in protocol with TDH2, the plaintiff submits $[u_i, e_i, f_i]$ such that $e_i = \mathrm{H}_2\left(u_i, \hat{u}_i, \hat{h}_i\right)$ where $\hat{u}_i = \frac{u^{f_i}}{{u_i}^{e_i}}$ and $\hat{h}_i = \frac{g^{f_i}}{{h_i}^{e_i}}$. In the protocol based on PVSS, the plaintiff submits $[s_i, \pi_{s_i}]$, where π_{s_i} is the NIZK proof that shows $\log_g pk_i = \log_{s_i} \hat{s}_i$. Even if the sender knows s_i, it is impossible to maliciously slash a trustee without the π_{s_i}, which only the corresponding trustee knows.

Deploying such a mechanism would require the smart contract to access the ciphertext of a transaction (*e.g.*, u or $\hat{s}_i$ is necessary to verify the submitted share).

9 Evaluation

We prototype F3B by using post-merge Ethereum [23] as the underlying blockchain and Dela [19] written in Go [28] as the secret-management committee for our evaluation. Remaining consistent with Ethereum's security assumptions, one epoch length lasts 6.4 minutes and the trustees forming the secret-management committee from validators for a given epoch are randomly selected. We instantiate our cryptographic primitives by using the Edward25519 elliptic curve with 128-bit security supported by Kyber [37], an advanced cryptographic library. We ran experiments on a server with 32GB of memory and 40 (2.1GHz) CPU cores. The network communication delay is simulated to be a fixed 100ms. We further discuss F3B integration with Ethereum in Section 10.

Table 1 Latency Overhead for Ethereum Blockchain.

	Latency Overhead varying SMC sizes			
	TDH2		PVSS	
Confirmations	64	128	64	128
8	0.164%	0.206%	0.160%	0.214%
16	0.082%	0.103%	0.080%	0.107%
32	0.041%	0.052%	0.040%	0.053%
64	**0.020%**	**0.026%**	**0.020%**	**0.027%**
128	0.010%	0.013%	0.010%	0.013%

Table 2 Storage overhead for two protocols with different Secret-management Committee sizes.

	Storage Overhead (bytes)	
Number of trustees	TDH2 Protocol	PVSS protocol
8	80	792
16	80	1568
32	80	3120
64	80	6224
128	80	12 432

9.1 Latency

In Figure 4, we present the breakdown latency of each step for both TDH2 and PVSS protocols after a transaction finality while varying the number of SMC trustees from 8 to 128 nodes: (a) shares preparation by trustees, and (b) key reconstruction, and (c) decryption and execution. In addition, we show the time needed for PVSS shares generation by the sender in purple of Figure 4. As discussed in Section 5.3, only (b) and (c) represent the overhead at the per-transaction level.

Recall that the overall transaction latency using F3B is $mL_b + L_r$ (Section 5.2). In post-Merge Ethereum, the block time is fixed to 12 seconds, *i.e.*, $L_b = 12$ [9], and, by official standard, a block requires 64 block confirmations (two epochs) to be "finalized", *i.e.*, $m = 64$ [22].

Figure 5 presents the end-to-end latency comparison between the baseline protocol (Section 5.1), a sender-only commit-and-reveal protocol, as presented in Strawman 1 (Section 3.1), and F3B's protocol – varying the size of the secret-management committee stated after the string "F3B-". With the new PoS consensus, finalizing any data in Ethereum requires $mL_b = 64 * 12 = 768$ seconds. The baseline protocol's total latency is 768 seconds, as it requires only one write to the blockchain. Recall that in the sender-based commit-and-reveal approach (Strawman I), the sender commits a hash to the blockchain, taking 768 seconds, then reveals the transaction in another 768 seconds, totaling 1536 seconds. This results in a 100% latency overhead compared to the baseline, as the two steps must be sequential: the hash must be finalized on the blockchain before the reveal transaction can be propagated. Submarine, a more advanced approach that conceals the smart contract address, requires three sequential transactions. The sender must publish these three transactions in order, with the blockchain finalizing each one before the next one can be sent, suffering a latency delay of $768 * 3 = 2304$ seconds or a 200% latency overhead compared to the baseline [39, 11].

Compared with F3B, the reveal phase (key-reconstruction step) does not require the sender to write any data onto the blockchain. Therefore, we emphasize a significant difference between F3B and other application-based commit-and-reveal approaches, where F3B requires sending only one transaction to the underlying blockchain. Figure 5 shows that our design brings a low-latency overhead of 197ms and 205ms for two protocols, equivalent to 0.026% and 0.027% for Ethereum (relative to the 768 seconds finality time), under an SMC size of 128.

We acknowledge that some Ethereum users may accept a lower confirmation number to accept a transaction, even though Ethereum officially requires 64 blocks [22]. Without loss of generality, we outline different confirmation numbers with F3B's latency overhead in Table 1.

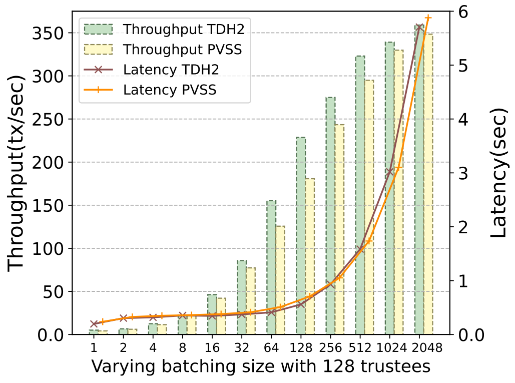


Figure 6 Performance of the key construction.

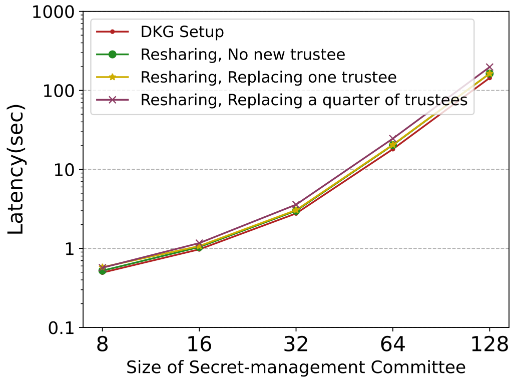


Figure 7 The latency cost of DKG setup and three resharing scenarios.

9.2 Throughput

Figure 6 presents the F3B's throughput results with a secret-management committee consisting of 128 trustees, assuming the underlying blockchain is not the bottleneck. If the keys are individually reconstructed, F3B provides limited throughput due to network transmission overhead incurred from sequential execution. Instead, we can batch keys by reconstructing them concurrently and presenting them in one network transmission. We present this batching effect in Figure 6 by varying the batching size to measure throughput and corresponding latency. By increasing the batching size from 1 to 2048, we can improve throughput from 5 txns/sec to 359 txns/sec with the TDH2 protocol, and from 4 txns/sec to 348 txns/sec with the PVSS protocol. The increased throughput comes with a higher latency cost: With a batching size of 2048, the key reconstruction step of TDH2 now takes 5.71 seconds to process, and the same step of PVSS takes 5.88 seconds; this latency is equivalent to a 0.74% and 0.77% latency overhead over Ethereum. Our results show that F3B provides more than sufficient throughput to support Ethereum (15 tx/sec [51]).

9.3 Reconfiguration in TDH2

Figure 7 demonstrates the cost of reconfiguring a secret-management committee in the TDH2 protocol. Recall that DKG is a one-time setup operation per epoch, bootstrapped during the previous epoch. Our experiment shows that, with a committee size of 128 trustees, DKG takes about 144 seconds; this is about 37.5% of Ethereum's epoch time (384 seconds). We offer a further discussion about the transition between two epochs in Section 10.1. To provide backward secrecy and dynamic membership, a secret-management committee can run a verifiable resharing protocol [62] within an epoch and, by keeping the public key, without interrupting users' encryption. Figure 7 illustrates the cost of three scenarios: (a) resharing among the same committee, (b) replacing one trustee, and (c) replacing a quarter of trustees. They all exhibit latency of the same magnitude.

9.4 Storage Overhead

In the TDH2 protocol, as the symmetric key is encrypted with the shared public key, the size of c_k is independent of the number of trustees. We also optimize the original TDH2 protocol to remove the label L from the ciphertext but only insert L in the computation and verification steps of each party (consensus group, secret-management committee, sender) for protection against replay attacks (Section 7.2). Ultimately, we achieve 80 bytes per transaction of the storage overhead presented in Table 2.

In the PVSS protocol, however, the ciphertext c_k contains encrypted shares, NIZK proofs, and polynomial commitments. The size of the c_k thus approximately grows linearly with the number of trustees, as demonstrated in Table 2. The difference in storage overhead is one of the trade-offs when system designers need to consider using which encryption algorithm in F3B, as discussed in Section 5.4.

10 Discussion

In this section, we discuss some deployment challenges. We leave a detailed analysis for future work.

10.1 Transition of Epoch

Each epoch has its unique public epoch key for a user to encrypt his symmetric key used for encrypting a transaction. However, users will have difficulty choosing the correct epoch key when the time is close to the transition between two epochs. With Fairblock [43] and Shutter [57, 58], undesirable transaction revealing occurs when a user chooses the wrong public key. Whereas, if the transaction is not finalized, F3B never reveals any transaction, regardless of the chosen key; thus offering confidentiality to all unfinalized transactions. We also expect that such an epoch transition is infrequent, compared with a block transition, thus it causes much less trouble to users. In F3B, if a user uses an old epoch key for his encryption, he can safely try again to select the new epoch key. To mitigate the issue even more, the expiring epoch committee can offer some grace period, thus allowing both old and new epoch keys to be valid for a certain period. This significantly reduces the danger of a user choosing an incorrect key.

10.2 Ethereum Gas Fees

Ethereum uses gas fees to cover the cost of executing a transaction and implements a maximum gas limit per block. Incorporating F3B on Ethereum would then require (1) that the gas limit of each transaction to be in cleartext, and (2) that the summation of all transactions' *gas limit* within a block does not exceed the *block gas limit*. This opens the possibility for another type of spamming attack (Section 8.1), where an adversary submits transactions with substantial gas limits, thus leaving little room for other transactions.

Recall that the actual gas used by transactions cannot be determined because the sender encrypts its contents, and that accurately estimating the gas cost of a transaction is particularly difficult due to the uncertainty of the global state when validators process the transaction. One potential approach to mitigating this kind of attack would be to then burn the remaining (unused) gas. However, this approach could be too strict in practice, hence we can instead envision partial refunds: refunding the remaining gas up to a percentage.

10.3 Verifiable Key Propagation

Under our proposed protocol, every consensus node must fetch the shares and run the Lagrange interpolation to reconstruct the key. Would it instead be possible for one of the consensus nodes to reconstruct the symmetric key k from the secret shares and to propagate it to other consensus nodes with a succinct proof?

Therefore, we propose a solution that requires additional storage overhead in exchange for faster verification: Instead of constructing their encrypted transaction as (c_k, c_{tx}), the sender additionally adds a hash of the symmetric key $h_k = H(k)$ as the third entry, creating the following signed write transaction: $\text{tx}_{\text{w}} = [c_k, c_{\text{tx}}, h_k]_{\text{sig}_{\text{sk}_A}}$.

During key reconstruction, after recovering or receiving k, consensus nodes need to verify whether the hash of k is consistent with the one (h_k) published on the ledger. If it is consistent, a consensus node can continue to decrypt the transaction and propagate the key k to others. If it is inconsistent, however, a consensus node must reconstruct the key from decryption shares and publish the shares to the underlying blockchain to slash the sender who provides a wrong h_k.

10.4 Metadata Leakage

In our architecture, adversaries can only observe encrypted transactions until they are finalized, thus preventing the revelation of transaction contents to launch front-running attacks. Nevertheless, to launch speculative attacks, adversaries can rely on side channels such as transaction metadata. Concretely, as the sender needs to pay the storage fee (Section 8.1) for publishing an encrypted transaction to the underlying blockchain, this leaks the sender's address. Knowledge of the sender's address can help in launching a front-running attack because an adversary might be able, based on the sender's history, to predict the sender's behavior. To prevent this second-order front-running attack, a sender can use a different address to pay for the storage fee. The underlying blockchain can also offer anonymous payment to users, such as Zerocash [49] or a mixing service [66], to further hide the payment address. Another side-channel leakage is the size of the encrypted transaction or the time the transaction is propagated. A possible remedy for mitigating metadata leakage is PURBs [45].

10.5 Key Storage and Node Catchup

In our protocol, if a new node wants to join the consensus group, it cannot execute the historical transactions to catch up, unless it obtains all decryption keys. The secret-management committee or consensus group can store these keys independently from the blockchain, but this requires them to maintain an additional immutable ledger. As consensus nodes already maintain one immutable storage medium, namely the underlying blockchain, the keys can be stored on this medium as metadata; and the blockchain rule can require storing valid keys when producing blocks.

However, this optimization brings about a timing issue, *i.e.*, When should the blockchain require the consensus group to store keys in a block? From our protocol, the transaction is finalized at block height n and revealed at block height $n + m$, thus making the earliest block to write the key at block height $n + m + 1$. With respect to the latest block height to write the key, there is much more flexibility and we need to consider the balance between the delay tolerance for all consensus nodes to retrieve the key and the time that consensus nodes must retain the key. Assuming that the key reconstruction step takes up to δ block times, the key should be written in or before the block $n + m + \delta$.

Although this setup would work well for a blockchain with fixed block time, care must be taken for blockchains where block time is probabilistic as the key might not have been replicated to all consensus nodes at block height $n + m + \delta$, thus some artificial delay for new blocks could be induced.

11 Related Work

Namecoin is a decentralized name service and an early work on front-running protection using a commit-and-reveal design [29]. In Namecoin, a user first broadcasts a salted hash of their name and then, after finality, broadcasts the actual name. Our first strawman protocol (Section 3.1) is based on Namecoin.

After Namecoin, Eskandari et al. [20] systematized front-running attacks on the blockchain by presenting three types of front-running attacks: displacement, insertion, and suppression. Daian et al. [15] also quantified front-running attacks from an economic point of view, determining that front-running attacks can also pose a security risk to the underlying consensus layer by incentivizing unnecessary forks driven by the maximal extractable value (MEV).

Many previous works explore the idea of applying threshold cryptography on blockchain. Virtual ASICs use threshold encryption to implement an all-or-nothing broadcast in the blockchain layer [26]. Sikka [59], Ferveo (Anoma) [5], Schmid [52], Dahlia [40], and Helix [3] apply threshold encryption to mitigate front-running but only present discussions with specific consensus algorithms. Fairblock [43] and Shutter [57, 58] enable encrypted transactions on a per-block basis, but if an encrypted transaction fails to be included in the sender-chosen block, then the transaction would be revealed; our Strawman III design (Section 3.3) is based on their approach.

Calypso is a framework that enables on-chain secrets by adopting threshold encryption governed by a secret-management committee [33]. Calypso allows ciphertexts to be stored on the blockchain and collectively decrypted by trustees according to a predefined policy. F3B leverages Calypso to specifically mitigate front-running attacks and extends its functionality to release the transaction contents once finalized automatically. F3B adopts per-transaction encryption, thus protecting all unfinalized transactions from front-running attacks, even if the transactions are delayed.

Other works adopt different approaches to mitigate front-running. A series of recent studies focus on fair ordering [30, 35, 36], but they cannot prevent an adversary with a rapid network connection [4]. Wendy explores the possibility of combining fair ordering with commit-and-reveal [36] but is in need of quantitative overhead analysis. Submarine is an application-layer front-running protection approach that extends a commit-and-reveal design to prevent leakage of the smart contract address. However, it presents a high latency overhead by requiring senders to have three rounds of communication with the underlying blockchain [39, 11].

Some works adopt time-lock puzzles [48] to blind transactions. For example, the injective protocol [12] uses a verifiable delay function [10] to achieve a proof-of-elapsed-time. However, an open challenge remains to link the time-lock puzzle parameters to an actual real-world delay [4].

Finally, works such as MEV-SGX [41], Tesseract [6], Secret Network [54], and Fairy [60] use a trusted execution environment [64] to mitigate front-running. Nevertheless, these approaches use a centralized component that is then subject to a single point of failure or compromise [47, 61].

12 Conclusion

In this paper, we have introduced F3B, a novel blockchain architecture that addresses front-running attacks with TDH2 and PVSS as threshold encryption protocols on a per-transaction basis. Our evaluation of F3B demonstrates that F3B is agnostic to consensus algorithms and to existing smart-contract implementations. We have also shown that F3B meets the necessary throughput while presenting a low-latency overhead, thus fitting with Ethereum. Given that the deployment of F3B would require modifications to a blockchain's execution layer, F3B, in return, would also provide a substantial benefit: the F3B-deployed blockchain would now, by default, contain standard front-running protection for all applications in need at once without requiring any modifications to smart contracts themselves.

References

1 Nasdaq: Front running. `https://www.nasdaq.com/glossary/f/front-running`, 2018. Accessed: 2022-04-17.

2 Prashant Ankalkoti and SG Santhosh. A relative study on bitcoin mining. *Imperial Journal of Interdisciplinary Research (IJIR)*, 3(5):1757–1761, 2017.

3 Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. Helix: A scalable and fair consensus algorithm resistant to ordering manipulation. *Cryptology ePrint Archive*, 2018.

4 Carsten Baum, James Hsin-yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. Sok: Mitigation of front-running in decentralized finance. *Cryptology ePrint Archive*, 2021.

5 Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in bft networks. *Cryptology ePrint Archive*, 2022.

6 Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1521–1538, 2019.

7 Dan Bernhardt and Bart Taub. Front-running dynamics. *Journal of Economic Theory*, 138(1):288–296, 2008.

8 Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 967–980, 2013.

9 Blocks, 2022. Accessed: 2022-10-03. URL: `https://ethereum.org/en/developers/docs/blocks`.

10 Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

11 Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. Enter the hydra: Towards principled bug bounties and {Exploit-Resistant} smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1335–1352, 2018.

12 Eric Chen and Albert Chon. Injective protocol: A collision resistant decentralized exchange protocol [White paper], 2018. URL: `https://coinpare.io/whitepaper/injective-protocol.pdf`.

13 Michele Ciampi, Muhammad Ishaq, Malik Magdon-Ismail, Rafail Ostrovsky, and Vassilis Zikas. Fairmm: A fast and frontrunning-resistant crypto market-maker. *Cryptology ePrint Archive*, 2021.

14 Coinbase. Coinbase confirmations, 2022(?). Accessed: 2022-03-03. URL: `https://help.coinbase.com/en/coinbase/getting-started/crypto-education/glossary/confirmations`.

15 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

16 N Neha Dalwadi and C Mamta Padole. Comparative study of clock synchronization algorithms in distributed systems. *Advances in Computational Sciences and Technology*, 10(6):1941–1952, 2017.

17 dapp.org. Uniswap v2 audit report, 2020. Accessed: 2022-01-22. URL: `https://dapp.org.uk/reports/uniswapv2.html`.

18 Defi pulse. URL: `https://www.defipulse.com/?time=All`.

19 Dedis ledger architecture. URL: `https://github.com/dedis/dela`.

20 Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: Front-running attacks on blockchain. In *Financial Cryptography Workshops*, volume 11599 of *Lecture Notes in Computer Science*, pages 170–189. Springer, 2019.

21 Gas and fees, 2022. Accessed: 2022-10-03. URL: `https://ethereum.org/en/developers/docs/gas/`.

22 Gasper, 2022. Accessed: 2022-10-03. URL: `https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/gasper/`.

23 The merge, 2022. Accessed: 2022-10-03. URL: `https://ethereum.org/en/upgrades/merge/`.

24 Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

25 Flashbots protect. URL: `https://docs.flashbots.net/flashbots-protect/overview`.

26 Chaya Ganesh, Claudio Orlandi, Daniel Tschudi, and Aviv Zohar. Virtual asics: generalized proof-of-stake mining in cryptocurrencies. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 173–191. Springer, 2021.

27 Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 295–310. Springer, 1999.

28 Go. The go programming language, 2009. URL: `https://go.dev`.

29 Harry A Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan. An empirical study of namecoin and lessons for decentralized namespace design. In *WEIS*. Citeseer, 2015.

30 Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. *Cryptology ePrint Archive*, 2021.

31 Olga Kharif. Cryptokitties mania overwhelms ethereum network's processing, 2017. URL: `https://www.bloomberg.com/news/articles/2017-12-04/cryptokitties-quickly-becomes-most-widely-used-ethereum-app`.

32 Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th usenix security symposium (usenix security 16)*, pages 279–296, 2016.

33 Eleftherios Kokoris-Kogias, Enis Ceyhun Alp, Linus Gasser, Philipp Jovanovic, Ewa Syta, and Bryan Ford. Calypso: Private data management for decentralized ledgers. *Cryptology ePrint Archive*, 2018.

34 Kraken. Cryptocurrency deposit processing times, 2022(?). Accessed: 2022-03-03. URL: `https://support.kraken.com/hc/en-us/articles/203325283-`.

35 Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 25–36, 2020.

36 Klaus Kursawe. Wendy grows up: More order fairness. In *International Conference on Financial Cryptography and Data Security*, pages 191–196. Springer, 2021.

37 `https://github.com/dedis/kyber`The Kyber Cryptography Library, 2010 – 2022.

38 Michael Lewis. *Flash boys: a Wall Street revolt.* WW Norton & Company, 2014.

39 LibSubmarine. Defeat front-running on ethereum, 2017(?). Accessed: 2022-01-24. URL: `https://libsubmarine.org`.

40 Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (mev) protection on a dag. *arXiv preprint arXiv:2208.00940*, 2022.

41 Mev-sgx: A sealed bid mev auction design, 2021. URL: `https://ethresear.ch/t/mev-sgx-a-sealed-bid-mev-auction-design/9677`.

42 Edvardas Mikalauskas. 280 million stolen per month from crypto transactions, 2021. Accessed: 2022-02-16. URL: `https://cybernews.com/crypto/flash-boys-2-0-front-runners-draining-280-million-per-month-from-crypto-transactions`.

43 Peyman Momeni. Fairblock: Preventing blockchain front-running with minimal overheads. Master's thesis, University of Waterloo, 2022.

44 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

45 Kirill Nikitin, Ludovic Barman, Wouter Lueks, Matthew Underwood, Jean-Pierre Hubaux, and Bryan Ford. Reducing metadata leakage from encrypted files and communication with purbs. *Proceedings on Privacy Enhancing Technologies*, 2019(4):6–33, 2019.

46 Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.

47 Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Crosstalk: Speculative data leaks across cores are real. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1852–1867. IEEE, 2021.

48 Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology. Laboratory for Computer Science, 1996.

49 Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014.

50 Alexander Savelyev. Contract law 2.0:'smart'contracts as the beginning of the end of classic contract law. *Information & communications technology law*, 26(2):116–134, 2017.

51 Markus Schäffer, Monika di Angelo, and Gernot Salzer. Performance and scalability of private ethereum blockchains. In *International Conference on Business Process Management*. Springer, 2019.

52 Noah Schmid. Secure causal atomic broadcast, 2021. URL: `https://crypto.unibe.ch/archive/theses/2021.bsc.noah.schmid.pdf`.

53 Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*, pages 148–164. Springer, 1999.

54 Secret markets: Front running prevention for automated market makers, 2020. URL: `https://scrt.network/blog/secret-markets-front-running-prevention`.

55 Adi Shamir. Identity-based cryptosystems and signature schemes. In *Workshop on the theory and application of cryptographic techniques*, pages 47–53. Springer, 1984.

56 Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–16. Springer, 1998.

57 Combating front-running and malicious mev using threshold cryptography. URL: `https://blog.shutter.network`.

58 Shutterized beacon chain. URL: `https://ethresear.ch/t/shutterized-beacon-chain/12249`.

59 URL: `https://sikka.tech`.

60 Chrysoula Stathakopoulou, Signe Rüsch, Marcus Brandenburger, and Marko Vukolić. Adding fairness to order: Preventing front-running attacks in bft protocols using tees. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 34–45. IEEE, 2021.

61 Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008, 2018.

62 Theodore M Wong, Chenxi Wang, and Jeannette M Wing. Verifiable secret redistribution for archive systems. In *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, pages 94–105. IEEE, 2002.

63 Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

64 Bin Cedric Xing, Mark Shanahan, and Rebekah Leslie-Hurd. Intel® software guard extensions (intel® SGX) software support for dynamic memory allocation inside an enclave. *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, pages 1–9, 2016.

65 Haoqian Zhang, Louis-Henri Merino, Vero Estrada-Galinanes, and Bryan Ford. F3b: A low-latency commit-and-reveal architecture to mitigate blockchain front-running. *arXiv preprint arXiv:2205.08529*, 2022.

66 Jan Henrik Ziegeldorf, Roman Matzutt, Martin Henze, Fred Grossmann, and Klaus Wehrle. Secure and anonymous decentralized bitcoin mixing. *Future Generation Computer Systems*, 80:448–466, 2018.

Designing Multidimensional Blockchain Fee Markets

Theo Diamandis
MIT CSAIL, Cambridge, MA, USA

Alex Evans
Bain Capital Crypto, San Francisco, CA, USA

Tarun Chitra
Gauntlet, New York, NY, USA

Guillermo Angeris
Bain Capital Crypto, San Francisco, CA, USA

Abstract

Public blockchains implement a fee mechanism to allocate scarce computational resources across competing transactions. Most existing fee market designs utilize a joint, fungible unit of account (*e.g.*, *gas* in Ethereum) to price otherwise non-fungible resources such as bandwidth, computation, and storage, by hardcoding their relative prices. Fixing the relative price of each resource in this way inhibits granular price discovery, limiting scalability and opening up the possibility of denial-of-service attacks. As a result, many prominent networks such as Ethereum and Solana have proposed multidimensional fee markets. In this paper, we provide a principled way to design fee markets that efficiently price multiple non-fungible resources. Starting from a loss function specified by the network designer, we show how to dynamically compute prices that align the network's incentives (to minimize the loss) with those of the users and miners (to maximize their welfare), even as demand for these resources changes. We derive an EIP-1559-like mechanism from first principles as an example. Our pricing mechanism follows from a natural decomposition of the network designer's problem into two parts that are related to each other via the resource prices. These results can be used to efficiently set fees in order to improve network performance.

2012 ACM Subject Classification Theory of computation → Convex optimization; Information systems → Digital cash; Theory of computation → Algorithmic mechanism design

Keywords and phrases Blockchains, transaction fees, convex optimization, mechanism design

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.4

Related Version *Full Version*: `https://arxiv.org/abs/2208.07919` [27]

Supplementary Material *Software (Source Code)*: `https://github.com/bcc-research/resource-pricing`, archived at `swh:1:dir:7fda6b4657a257cb9fd944628b905f45ec938c09`

Acknowledgements We would like to thank John Adler, Vitalik Buterin, Dev Ojha, Kshitij Kulkarni, Matheus Ferreira, Barnabé Monnot, and Dinesh Pinto for helpful conversations, insights, and edits. We're especially appreciative to John Adler for bearing with us through many drafts of this work and consistently providing valuable feedback.

1 Introduction

Public blockchains allow any user to submit a *transaction* that modifies the shared state of the network. Transactions are independently verified and executed by a decentralized network of *full nodes*. Because full nodes have finite resources, blockchains limit the total computational resources that can be consumed per unit of time. As user demand may fluctuate, most blockchains implement a *transaction fee* mechanism in order to allocate finite computational capacity among competing transactions.

Smart contracts and gas. Many blockchains allow transactions to submit and/or execute programs that exist on-chain called *smart contracts*. Once such a transaction is included in a block, full nodes must re-execute the transaction in order to obtain the resulting updated state of the ledger. All of these transactions consume computational resources, whose total supply is finite. To prevent transactions with excessive resource use and transaction spam, some smart-contract blockchains require users to pay fees in order to compensate the network for processing their transactions.

Most smart contract platforms calculate transaction fees based on a joint unit of account. In the Ethereum Virtual Machine (EVM), this unit is called *gas*. Each operation in the EVM requires a hardcoded amount of gas which is intended to reflect its relative resource usage. The network enforces a limit on the total amount of gas consumed across all transactions in a block. This limit, called the *block limit*, prevents the chain from expending computational resources too quickly for full nodes to catch up to the latest state in a reasonable amount of time. Block limits must take into account the maximum amount of each resource that each block may consume (such as storage, bandwidth, or memory) without posing an extreme burden on full nodes meeting the minimal hardware specifications. Because the block limit fixes the total gas supply in each block, the price of gas in "real" terms (*e.g.*, in terms of US Dollars) fluctuates based on demand for transactions in the block.

One-dimensional transaction fees. Calculating transaction fees through a single, joint unit of account, such as gas, introduces two major challenges. First, if the hardcoded costs of each operation are not precisely reflective of their relative resource usage, there is a possibility of denial-of-service attacks (specifically, *resource exhaustion attacks* [44]), where an attacker exploits resource mispricing to overload the network. Historically, the Ethereum network has suffered from multiple DoS attacks [13, 14, 52] and has had to manually adjust the relative prices accordingly (*e.g.*, [12, 22]). Amending the hardcoded costs of each gas operation in response to such attacks typically requires a hard fork of the client software.

Second, one-dimensional fee markets limit the theoretical throughput of the network. Using a joint unit of account, such as gas, to price separate resources decouples their price from supply and demand. As an extreme example to illustrate this dynamic, if the block gas limit is fully saturated with exclusively CPU-intensive operations, gas price will increase as transactions compete for limited space. The cost of transactions that consume exclusively network bandwidth (and nearly no CPU resources) will also increase because these also require gas, even if demand and supply for bandwidth resources across the network remain unchanged. As a result, fewer bandwidth-intensive transactions can be included in the block despite spare capacity, limiting throughput. This limitation occurs because the joint unit of account only allows the network to price resources *relative to each other* and not in real terms based on the supply and demand for each resource. As we will soon discuss, allowing resources to be priced separately promotes more efficient resource utilization by enabling more precise price discovery. We note that this increase in throughput need not increase hardware requirements for full nodes.

Multidimensional fee markets. Due to the potential scalability benefits of more granular price discovery, a number of popular smart contract platforms are actively exploring multidimensional fee market mechanisms [5, 18]. We discuss some example proposals that are under active development below.

Rollups and data markets. Rollups are a popular scaling technology that effectively decouples transaction validation and execution from data and consensus [17]. In rollups, raw transaction data is posted to a base blockchain. Rollups also periodically post succinct proofs of valid execution to the base chain in order to enable secure bridging, prevent rollbacks, and arbitrate potential disputes by using the base chain as an anchor of trust. Rollups naturally create two separate fee markets, one for base layer transactions and one for rollup execution. As rollups have become a popular design pattern for achieving scalability, specialized blockchains (called *lazy blockchains*) that exclusively order raw data through consensus (*i.e.*, do not perform execution) have emerged [8, 7, 45, 43, 51]. These blockchains naturally allow for transaction data/bandwidth and execution to be priced through independent (usually one-dimensional) fee markets [4]. Similarly, Ethereum developers have proposed EIP-2242, wherein users may submit special transactions which contain an additional piece of data called a *blob* [2, 1]. Blobs may contain arbitrary data intended to be interpreted and executed by rollups rather than the base chain. Later, EIP-4844 extended these ideas by establishing a two-dimensional fee market wherein data blobs and base-chain gas have different limits and are priced separately [19]. EIP-4844 therefore intends to increase scalability for rollups, as blobs do not have to compete with base-chain execution.

Incentivizing parallelization. Most smart contract platforms, including the EVM, execute program operations sequentially by default, limiting performance. There are several proposals to enable parallel execution in the EVM which generally fall into two categories. The first involves minimal changes to the EVM and pushes the responsibility of identifying opportunities for parallel execution to full nodes [33, 24, 49]. The other approach involves *access lists* which require users to specify which accounts their transaction will interact with, allowing the network to easily identify non-conflicting transactions that can be executed in parallel [15, 23]. While Ethereum makes access lists optional, other virtual machine implementations, such as Solana Sealevel and FuelVM, require users to specify the accounts their transaction will interact with [53, 38, 3]. Despite this capability, a large fraction of transactions often want to access the same accounts in scenarios including auctions, arbitrage opportunities, and new product launches. Such contention significantly limits the potential benefits of the virtual machine's parallelization capabilities. As a result, developers of Solana proposed multidimensional fee markets that price interactions with each account separately [54]. Transactions which require sequential execution are charged higher fees, incentivizing usage of spare capacity on multi-core machines.

This paper. In this paper, we formally illustrate how to efficiently update resource prices, what optimization problem these updates attempt to solve, and some consequences of these observations. We frame the pricing problem in terms of an idealized, omniscient network designer who chooses transactions to include in blocks in order to maximize total welfare, subject to demand constraints. (The designer is omniscient as the welfare is unknown and likely unmeasurable in any practical setting.) We show that this problem, which is the "ideal end state" of a blockchain but not immediately useful by itself, decomposes into two problems, coupled by the resource prices. One of these two problems represents the cost to the network for providing certain resources and is simple enough to be solved on chain. The other is a maximal-utility problem that miners and users implicitly solve when creating and including transactions for a given block. Correctly setting the resource prices aligns incentives such that the resource costs to the network are exactly balanced by the utility gained by the users and miners. This, in turn, leads to block allocations which solve the original "ideal" problem, on average.

For convenience, we provide appendix A in the full version [27] as a short introduction to convex optimization. We recommend readers unfamiliar with convex optimization at least skim this appendix, as it provides a short introduction to all the mathematical definitions and major theorems used in this paper.

1.1 Related work

The resource allocation problem has been studied in many fields, including operations research and computer systems. Agrawal, et al. [6] proposed a similar formulation and price update scheme for fungible resources where utility is defined per-transaction. Prior work on blockchain transaction fees varies from the formal axiomatic analysis of game theoretic properties that different fee markets should have [48, 25] to analysis of dynamic fees from a direct algorithmic perspective [31, 39, 47]. Works of the latter form generally focus on whether the system converges to an equilibrium. Moreover, these mechanisms focus on dynamic pricing at the block level (*e.g.*, how many transactions should be allowed in a block?) and not directly on questions of how capacity should be allocated and priced across different transaction types.

EIP-1559. EIP-1559 [21], implemented last year, proposed major changes to Ethereum's transaction fee mechanism. Specifically, EIP-1559 implemented a base fee for transactions to be included in each block, which is dynamically adjusted to hit a target block usage and burned instead of rewarded to the miners. While EIP-1559 is closely related to the problem we consider, it ultimately has a different goal: EIP-1559 attempts to make the fee estimation problem easier in a way that disincentivizes manipulation and collusion [16, 48]. We, on the other hand, aim to price resources dynamically to achieve a given network-specified objective. Finally, we note that prior work such as [31] has proved incentive compatibility for a large set of mechanisms that are a superset of EIP-1559. It is likely (but not proven in this work) that our model fits within an extension of their incentive compatibility framework. We leave game theoretic analysis and strategies to ensure incentive compatibility as future work.

2 Transactions and resources

Before introducing the network's resource pricing problem, we discuss the general set up and motivation for the problem in the case of blockchains.

Transactions. We will start by reasoning about *transactions*. A transaction can represent arbitrary data sent over the peer-to-peer network in order to be appended to the chain. Typically, a transaction will represent a value transfer or a call to a smart contract that exists on chain. These transactions are broadcasted by users through the peer-to-peer network and collected by consensus nodes in the *mempool*, which contains all submitted transactions that have not been included on chain. A *miner* gets to choose which transactions from the mempool are included on chain. Miners may also outsource this process to a block builder in exchange for a reward [20]. Once a transaction is included on chain, it is removed from the mempool. (Any conflicting transactions are also removed from the mempool.)

Nodes. Every transaction needs to be executed by *full nodes* (which we will refer to as *nodes*). Nodes compute the current state of the chain by executing and checking the validity of all transactions that have been included on the chain since the genesis block. Many blockchains have minimum computational requirements for nodes in a blockchain: any node meeting these requirements should be able to eventually "catch up" to the head of the chain in a reasonable amount of time, *i.e.*, execute all transactions and reach the latest state, even as the chain continues to grow. (For example, Ethereum requires 4GB RAM and 2TB of SSD Storage, and they recommend at least a Intel NUC, 7th gen processor [30].) These requirements both limit the computational resources each block is allowed to consume and promote decentralization by ensuring the required hardware does not become prohibitively expensive. If transactions are included in a blockchain faster than nodes are able to execute them, nodes cannot reconstruct the latest state and can't ascertain the validity of the chain. This type of denial of service (DoS) attack is also referred to as a resource exhaustion attack. (As a side note, in some systems, it is possible to provide an easily-verifiable certificate that the state is correct. This certificate allows nodes to validate the state of the chain without executing all past transactions. In these systems, the time-consuming step is generating the certificate. A similar market mechanism might make sense for these systems, but we do not explore this topic here.)

Resource targets and limits. There are several ways to prevent this type of denial of service attack. For example, one method is to enforce that any valid transaction (or sequence of transactions, *e.g.*, a block) consumes at most some fixed upper bound of resources, or combinations of resources. These limits are set so that a node satisfying the minimum node requirements is always able to process transactions quickly enough to catch up to the head of the chain in a reasonable amount of time. Another possibility is to disincentivize miners and users from repeatedly including transactions that consume large amounts of resources while allowing short "bursts" of resource-heavy transactions. This margin needs to be carefully balanced so that a node meeting the minimum requirements is able to catch up after a certain period of time. This intuition suggests having both a "resource target" and a larger "resource limit," which we will formalize in what follows.

Resources. Most blockchain implementations define a number of *meterable resources* (simply *resources* from here on out) which we will label $i = 1, \dots, m$, that a transaction can consume. For example, in Ethereum, the resources could be the individual Ethereum Virtual Machine (EVM) opcodes used in the execution of a transaction. In this paper, the notion of a resource is much more general than simply an opcode or an "execution unit". Here, a resource can refer to anything as coarse as "total bandwidth usage" or as granular as individual instructions executed on a specific core of a machine – the only requirement for a resource, as used in this paper, is that it can be easily and consistently metered across any node. For a given transaction $j = 1, \dots, n$, we will let $a_j \in \mathbb{R}^m_+$ be the vector of resources that transaction j consumes. In particular, the ith entry of this vector, $(a_j)_i$, denotes the amount of resource i that transaction j uses. We note that the resource usage $(a_j)_i$ does not, in fact, need to be nonnegative. While our mechanism works in the more general case (with some small modifications), we assume nonnegativity in this work for simplicity.

Combined resources. This framework naturally includes combinations of resources as well. For example, we may have two resources R_1 and R_2, each cheap to execute once in a transaction, which are costly to execute serially (*i.e.*, it is costly to execute R_1 and then R_2 in the same transaction). In this case, we can create a "combined" resource R_1R_2 which is itself metered separately. Note that, in our discussion of resources, there is no requirement that the resources be independent in any sense, and such "combined resources" are themselves very reasonable to consider.

Resource utilization targets. As mentioned previously, many networks have a minimum node requirement, implying a sustained target for resource utilization in each group of transactions added to the blockchain. (For simplicity, we will call this sequence of transactions a *block*, though it could be any collection of transactions that makes sense for a given blockchain.) We will write this resource utilization target as a vector $b^\star \in \mathbb{R}^m$ whose ith entry denotes the desired amount of resource i to be consumed in a block. The resource utilization of a particular block is a linear function of the transactions included in a block. We will denote the included transactions as a Boolean vector $x \in \{0,1\}^n$ whose jth entry is one if transaction j is included in the block and is zero otherwise. We will write $A \in \mathbb{R}^{m \times n}$ as a matrix whose jth column is the vector of resources a_j consumed by transaction j. We can then write the total quantity of consumed resources by this block as

$$y = Ax, \tag{1}$$

where $y \in \mathbb{R}^m$ is a vector whose ith entry denotes the quantity of resource i consumed by all transactions included in the block. Additionally, we can write the deviation from the target utilization, sometimes called the residual, as

$$Ax - b^\star,$$

i.e., a vector whose ith element is the total quantity of resource i, consumed by transactions in this block, minus the target $b_i^\star$ for resource i. For example, in Ethereum post EIP-1559, there is only one resource, gas, which has a target of 15M [29]. (We will see later how this notion of a resource utilization target can be generalized to a loss function.)

Resource utilization limits. In addition to resource targets, a blockchain may introduce a resource limit $b \in \mathbb{R}^m$ such that any valid block with transactions x must satisfy

$$Ax \leq b.$$

Continuing the example from before, Ethereum after EIP-1559 has a single resource, gas, with a resource limit of 30M.

Network fees. We want to incentivize users and miners to keep the total resource usage "close" to $b^\star$. To this end, we introduce a *network fee* $p_i \in \mathbb{R}$ for resource $i = 1, \dots, m$, which we will sometimes call the *resource price*, or just the *price*. If transaction j with resource vector a_j is included in a block, a fee $p^T a_j = \sum_i p_i (a_j)_i$ must be paid to the network. (In Ethereum, the network fee is implemented by burning some amount of the gas fee for each transaction and can be thought of as the "burn rate".) For now, we will assume that as the fee p_i gets larger, the amount of resource i used in a block $(Ax)_i$ will become smaller and vice versa.

Resource mispricing. Given A and b, it is, in general, not clear how to set the fees p in order to ensure the network has good performance (in other words, so that the resource utilization is "close" to $b^\star$). As a real world example, starting in Ethereum block number 2283416 (Sept. 18, 2016) an attacker exploited the fact that resources were mispriced for the EXTCODESIZE opcode, causing the network to slow down meaningfully. This mispricing was fixed via the hard fork on block 2463000 (Oct. 18, 2016) with details outlined in EIP-150 [12]. (The effects of this mispricing can still be observed when attempting to synchronize a full node. A dramatic slowdown in downloading and processing blocks happens starting at the previously mentioned block height.) Though usually less drastic, there have been similar events on other blockchains, underscoring the importance of correctly setting resource prices.

Setting fees. We want a simple update rule for the network fees p with the following properties:

1. If $Ax = b^\star$, then there is no update.
2. If $(Ax)_i > b^\star_i$, then p_i increases.
3. Otherwise, if $(Ax)_i < b^\star_i$, then p_i decreases.

There are many update rules with these properties. As a simple example, we can update the network fees as

$$p^{k+1} = \left(p^k + \eta(Ax - b^\star)\right)_+, \tag{2}$$

where $\eta > 0$ is some (usually small) positive number, often referred to as the "step size" or "learning rate", k is the block number such that p^k are the resource prices at block k, and $(w)_+ = \max\{0, w\}$ for scalar w and is applied elementwise for vectors. Recently, Ethereum developers [18] proposed the update rule

$$p^{k+1} = p^k \odot \exp\left(\eta(Ax - b^\star)\right). \tag{3}$$

Here, $\exp(\cdot)$ is understood to apply elementwise, while $\odot$ is the elementwise or Hadamard product between two vectors. The remainder of this paper will show that many update rules are attempting to (approximately) solve an instance of a particular optimization problem with a natural interpretation, where parts of the update rule come from a specific choice of objective function by the network designer. (We show in section 3.3 that a similar rule to (3) can be derived as a consequence of the framework presented in this paper, under a particular choice of variable transformation.)

We note that [31] has studied fixed points of such iterations in the one-dimensional case, extending the analysis of EIP-1559. However, the multidimensional scenario can be quite a bit more subtle to analyze. For instance, the multiplicative update rule (3) can admit "vanishing gradient" behavior in high-dimensions [34]. We suspect that the one-dimensional fee model of [31] can be extended to the multidimensional rules (2) and (3) but leave this for future work.

3 Resource allocation problem

As system designers, our ultimate goal is to maximize utility of the underlying blockchain network by appropriately allocating resources to transactions. However, we cannot perform this allocation directly, since we cannot control what users and miners wish to include in blocks, nor do we know what the utility of a transaction is to users and miners. Instead, we aim to set the fees p to ensure that the resource usage is approximately equal to the desired target, which we will represent as an objective function. We will show later that the update mechanisms proposed above naturally fall out of a more general optimization formulation. Similarly to the Transmission Control Protocol (TCP), each update rule is a result of a particular objective function, chosen by the network designer [42, 41].

Loss function. We define a *loss function* of the resource usage, $\ell : \mathbb{R}^m \to \mathbb{R} \cup \{\infty\}$, which maps a block's resource utilization, y, to the "unhappiness" of the network designer, $\ell(y)$. We assume only that ℓ is convex and lower semicontinuous. (We will not require monotonicity, nonnegativity, or other assumptions on ℓ, but we will show that useful properties do hold in these scenarios.)

For example, the loss function can encode "infinite dissatisfaction" if the resource target is violated at all:

$$\ell(y) = \begin{cases} 0 & y = b^\star \\ \infty & \text{otherwise.} \end{cases} \tag{4}$$

(Functions of this form, which are either 0 or ∞ at every point, are known as *indicator functions.*) Note also that this loss is not differentiable anywhere, but it is convex. Another possible loss, which is also an indicator function, is

$$\ell(y) = \begin{cases} 0 & y \le b^\star \\ \infty & \text{otherwise.} \end{cases} \tag{5}$$

This loss can roughly be interpreted as: we do not mind any usage below $b^\star$, but we are infinitely unhappy with any usage above the target amounts. Alternatively, we may only care about large deviations from the target $b^\star$:

$$\ell(y) = (1/2)\|y - b^\star\|_2^2,$$

or, perhaps, require that the loss is simply linear and independent of $b^\star$,

$$\ell(y) = u^T y, \tag{6}$$

for some fixed vector $u \in \mathbb{R}^m$. Another important family of losses are those which are separable and depend only on the individual resource utilizations,

$$\ell(y) = \sum_{i=1}^{m} \phi_i(y_i) \tag{7}$$

where $\phi_i : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$ for $i = 1, \dots, m$, are convex, nondecreasing functions. (The loss (5) is a special case of this loss, while (6) is a special case when the vector u is nonnegative.) We will make the technical assumption that $\phi_i(0) < \infty$ for every i, otherwise no resource allocation would have finite loss.

We will see that each definition of a loss function implies a particular update rule for the network fees p. This utility function can more generally be engineered to appropriately capture tradeoffs in increasing throughput of a particular resource at the possible detriment of other resources.

Resource constraints. Now that we have defined the network designer's loss, which is a way of quantifying "unhappiness" when the resource usage is y, we need some way to define the transactions that users are willing to create and, importantly, that miners are willing (and able) to include. We do this in a very general way by letting $S \subseteq \{0, 1\}^n$ be the set of possible transactions that users and miners are willing and able to create and include. Note that this set is discrete and can be very complicated or potentially hard to maximize over (as is the case in practice). For example, the set S could encode a demand for transactions which depend on other transactions being included in the block (as is the case in, *e.g.*, miner extractable value [37, 26, 46]), network-defined hard constraints of certain resources (such as $Ax \le b$ for every $x \in S$), and even very complicated interactions among different transactions (if certain contracts can, for example, only be called a fixed number of times, as in NFT mints). We make no assumptions about the structure of this set S but only require that the included transactions, $x \in \{0, 1\}^n$, obey the constraint $x \in S$.

Convex hull of resource constraints. A network designer may be more interested in the long-term resource utilization of the network than the resource utilization of any one particular block. In this case, the designer may choose to "average out" each transaction over a number of blocks instead of deciding whether or not to include it in the next block. To that end, we, as designers, will be allowed to choose convex combinations of elements of the constraint set S, which we will write as $\mathbf{conv}(S)$. (In general, this means that we can pick probability distributions over the elements of S, and x is allowed to be the expectation of any such probability distribution; *i.e.*, we only require that, for the designer, x is reasonable "in expectation".) Here, components of x may vary continuously between 0 and 1; these values have a simple interpretation. If x_i is not 0 or 1, then we can interpret the quantity $1/x_i$ as roughly the number of blocks after which transaction i is included. Of course, neither users nor miners can choose transactions to be "partially included", so this property will only apply to the idealized problem we present below. While this relaxation might seem unrealistically "loose", we will see later how this easily translates to the realistic case where transactions are either included or not (that is, x_i is either 0 or 1) by users and miners.

Transaction utility. Finally, we introduce the *transaction utilities*, which we will write as $q \in \mathbb{R}^n$. The transaction utility q_j for transaction $j = 1, \dots, n$ denotes the users' and miners' joint utility of including transaction j in a given block. Note that it is very rare (if at all possible) to know the values of q. However, we will see that, under mild assumptions, we do not need to know the values of q in order to get reasonable prices, and reasonable update rules will not depend on q.

Resource allocation problem. We are now ready to write the *resource allocation problem*, which is to maximize the utility of the included transactions, minus the loss, over the convex hull of possible transactions:

$$\begin{array}{ll} \text{maximize} & q^T x - \ell(y) \\ \text{subject to} & y = Ax \\ & x \in \mathbf{conv}(S). \end{array} \tag{8}$$

This problem has variables $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$, and the problem data are the resource matrix $A \in \mathbb{R}^{m \times n}$, the set of possible transactions $S \subseteq \{0,1\}^n$, and the transaction utilities $q \in \mathbb{R}^n$. Because the objective function is concave and the constraints are all convex, this is a convex optimization problem. On the other hand, even though the set $\mathbf{conv}(S)$ is convex, it is possible that $\mathbf{conv}(S)$ does not admit an efficient representation (for example, it may contain exponentially many constraints) which means that solving this problem is, in general, not easy.

Interpretation. We can interpret the resource allocation problem (8) as the "best case scenario", where the designer is able to choose which transactions are included (or "partially included") in a block in order to maximize the total utility. While this problem is not terribly useful by itself, since (a) it cannot really be implemented in practice, (b) we often don't know q, and (c) we cannot "partially include" a transaction within a block, we will see that it will decompose naturally into two problems. One of these problems can be easily solved on chain, while the other is solved implicitly by the users (who send transactions to be included) and miners (who choose which transactions to include). The solutions to the latter problem can always be assumed to be integral; *i.e.*, no transactions are "partially included". This will allow us to construct a simple update rule for the prices, which does not depend on q. For the remainder of the paper, we will call this combination of users and miners the *transaction producers*.

Offchain agreements and producers. Due to the inevitability of user-miner collusion, we consider the combination of the two, the transaction producers, as the natural unit. For example, it is not easily possible to create a transaction mechanism where the users are forced to pay miners some fixed amount, since it is always possible for miners to refund users via some off-chain agreement [48]. Similarly, we cannot force miners to accept certain transactions from users, since a miner always has plausible deniability of not having seen a given transaction in the mempool. While not perfect for a general analysis of incentives, this conglomerate captures the dynamics between the network's incentives and those of the miners and users better than assuming each is purely selfishly maximizing their own utility (as opposed to strategically colluding) and suffices for our purposes.

3.1 Setting prices using duality

In this section, we will show a decomposition method for this problem. This decomposition method suggests an algorithm (presented later) for iteratively updating fees in order to maximize the transaction producers' utility minus the loss of the network, given historical observations.

To start, we will reformulate (8) slightly by pulling the constraint $x \in \mathbf{conv}(S)$ into the objective,

$$\begin{array}{ll} \text{maximize} & q^T x - \ell(y) - I(x) \\ \text{subject to} & y = Ax \end{array} \tag{9}$$

where $I : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is the indicator function defined as $I(x) = 0$ if $x \in \mathbf{conv}(S)$ and $I(x) = +\infty$ otherwise.

Dual function. The Lagrangian [11, §5.1.1] for problem (9) is then

$$L(x, y, p) = q^T x - \ell(y) - I(x) + p^T(y - Ax),$$

with dual variable $p \in \mathbb{R}^m$. This corresponds to "relaxing" the constraint $y = Ax$ to a penalty $p^T(y - Ax)$ assigned to the objective, where the price per unit violation of constraint i is p_i. (Negative values denote refunds.) Rearranging slightly, we can write

$$L(x, y, p) = p^T y - \ell(y) + (q - A^T p)^T x - I(x).$$

The corresponding dual function [11, §5.1.2], which we will write as $g : \mathbb{R}^m \to \mathbb{R} \cup \{-\infty\}$, is found by partially maximizing L over x and y:

$$g(p) = \sup_y \left(p^T y - \ell(y)\right) + \sup_x \left((q - A^T p)^T x - I(x)\right). \tag{10}$$

Discussion. The first term can be recognized as the Fenchel conjugate of ℓ [11, §3.3] evaluated at p, which we will write as $\ell^*(p)$, while the second term is the optimal value of the following problem:

$$\begin{array}{ll} \text{maximize} & (q - A^T p)^T x \\ \text{subject to} & x \in \mathbf{conv}(S), \end{array} \tag{11}$$

with variable $x \in \mathbb{R}^n$. We can interpret this problem as the transaction producers' problem of creating and choosing transactions to be included in a block in order to maximize their utility, after netting the fees paid to the network. (We will sometimes refer to this problem as the transaction producers' *packing problem.*) We note that the optimal value of (11) in terms of p, which we will write as $f(p)$, is the pointwise supremum of a family of linear (and therefore convex) functions of p, so it, too, is a convex function [11, §3.2.3]. Note that since the objective is linear, problem (11) has the same optimal objective value as the nonconvex problem

$$\begin{array}{ll} \text{maximize} & (q - A^T p)^T x \\ \text{subject to} & x \in S, \end{array}$$

where we have replaced $\mathbf{conv}(S)$ with S. (See full version [27] appendix A.2 for a simple proof.) Finally, since $f(p)$ is the optimal value of problem (11) for fees p, the dual function g can be written as

$$g(p) = \underbrace{\ell^*(p)}_{\text{network}} + \underbrace{f(p)}_{\text{tx producers}}.$$

Since the dual function g is the sum of convex functions ℓ^* and f, it, too, is convex. (We will make use of this property soon.) Having defined the dual function g, we will see how this function can give us a criterion for how to best set the network fees p.

Duality. An important consequence of the definition of the dual function g is *weak duality* [11, §5.2.2]. Specifically, letting $s^\star$ be the optimal objective value for problem (8), we have that $g(p) \geq s^\star$ for every possible choice of price $p \in \mathbb{R}^m$. This is true because we have essentially "relaxed" the constraint to a penalty, so any feasible point x, y for the original problem (9) always has 0 penalty. (There may, of course, be other points that are not feasible for (9) but are perfectly reasonable for this "relaxed" version, so we've only made the set of possibilities larger.) The proof is a single line:

$$g(p) = \sup_{x,y} L(x,y,p) \geq \sup_{y=Ax} L(x,y,p) = \sup_{y=Ax} \left(q^T x - \ell(y) - I(x)\right) = s^\star.$$

A deep and important result in convex optimization is that, in fact, there exists a $p^\star$ for which

$$g(p^\star) = s^\star,$$

under some basic constraint qualifications.[1] In other words, adding the constraint $y = Ax$ to the problem is identical to correctly setting the prices p. Since we know for any p that $g(p) \geq s^\star$ then

$$g(p^\star) = \inf_p g(p),$$

or, that $p^\star$ is a minimizer of g. This motivates an optimization problem for finding the prices.

[1] The condition is that the relative interior of $A\,\mathbf{conv}(S) \cap \mathbf{dom}\,\ell$ is nonempty. Here, we write $A\,\mathbf{conv}(S) = \{Ax \mid x \in \mathbf{conv}(S)\}$ and $\mathbf{dom}\,\ell = \{x \mid \ell(x) < \infty\}$, while the relative interior is taken with respect to the affine hull of the set. This condition almost always holds in practice for reasonable functions ℓ and sets S.

The dual problem. The *dual problem* is to minimize g, as a function of the fees p. In other words, the dual problem is to find the optimal value of

$$\text{minimize} \quad g(p), \tag{12}$$

with variable $p \in \mathbb{R}^m$. If we can easily evaluate g, then, since this problem is a convex optimization problem, as g is convex, solving it is usually also easy. An optimizer of the dual problem has a simple interpretation using its optimality conditions. Let $p^\star$ be a solution to the dual problem (12) for what follows. If the packing problem (11) has a unique solution $x^\star$ for $p^\star$, then the objective value f is differentiable at $p^\star$. (See full version [27] appendix A.2.) Similarly, under mild conditions on the loss function ℓ (such as strict convexity) the function ℓ^* is differentiable at $y^\star$, with derivative satisfying $\nabla\ell(y^\star) = p^\star$. In this case, the optimality conditions for problem (12) are that

$$\nabla g(p^\star) = y^\star - Ax^\star = 0. \tag{13}$$

In other words, the fees $p^\star$ that minimize (12) are those that charge the transaction producers the exact marginal costs faced by the network, $\nabla\ell(Ax^\star) = p^\star$. Furthermore, these are exactly the fees which incentivize transaction producers to include transactions that maximize the welfare generated minus the network loss, subject to resource constraints, since $y^\star$ and $x^\star$ are feasible and optimal for problem (8).

Differentiability. In general, g is not always differentiable, but is almost universally subdifferentiable, under mild additional conditions on ℓ (*e.g.*, ℓ does not contain a line). Condition (13) may then be replaced with

$$0 \in \partial g(p^\star) = -Y^\star(p^\star) + AX^\star(p^\star),$$

where

$$Y^\star(p) = \operatorname*{argmax}_y \left(p^T y - \ell(y)\right),$$

while $X^\star(p) \subseteq \mathbf{conv}(S)$ are the maximizers of problem (11) for price p. We define $AX^\star(p) = \{Ax \mid x \in X^\star(p)\}$, and write $\partial g(p^\star)$ for the subgradients of g at $p^\star$ (*cf.*, full version [27] appendix A). The condition then says that the intersection of the extremizing sets $Y^\star(p^\star)$ and $AX^\star(p^\star)$ is nonempty at the optimal prices $p^\star$. We show a special case of this below, when $p = 0$, with a direct proof using strong duality that does not require these additional conditions.

3.2 Properties

There are a number of properties of the prices p that can be derived from the dual problem (12).

Nonnegative prices. If the objective function ℓ is separable and nondecreasing, as in (7), then any price p_i feasible for problem (12) must be nonnegative, $p_i \geq 0$. (By feasible, we mean that $g(p) < \infty$.) To see this, note that, by definition (7), we have

$$\ell^*(p) = \sup_y \left(p^T y - \ell(y)\right) = \sum_{i=1}^m \sup_{y_i} \left(p_i y_i - \phi_i(y_i)\right),$$

so we can consider each term individually. If $p_i < 0$ then any $y_i \leq 0$ must have

$$p_i y_i - \phi_i(y_i) \geq p_i y_i - \phi_i(0) \to \infty,$$

as $y_i \to -\infty$ since $\phi_i(y_i)$ is nondecreasing in y_i. So $g(p) \to \infty$ and therefore this choice of p cannot be feasible, so we must have that $p_i \geq 0$.

Superlinear separable losses. If the losses ϕ_i are superlinear, in that

$$\frac{\phi_i(z)}{z} \to \infty, \tag{14}$$

as $z \to \infty$ and bounded from below, in addition to being nondecreasing, then $\ell^*(p)$ is finite for $p \geq 0$. This means that the effective domain of g, defined as the set of prices for which g is finite,

$$\mathbf{dom}\, g = \{p \in \mathbb{R}^m \mid g(p) < \infty\},$$

is exactly the nonnegative orthant. (This discussion may appear somewhat theoretical, but we will see later that it turns out to be an important practical point when updating prices.) While not all losses are superlinear, we can always make them so by, *e.g.*, adding a small, nonnegative squared term to ϕ_i, say

$$\tilde{\phi}_i(z) = \phi_i(z) + \rho(z)_+^2,$$

where $(z)_+ = \max\{0, z\}$ and $\rho > 0$, or by setting $\phi_i(z) = \infty$ for $z \geq 0$ large enough.

Subsidies. Alternatively, if the function ℓ is decreasing somewhere on the interior of its domain, then there exist points $y^\star$ for which prices p_i are negative – *i.e.*, sometimes the network is willing to subsidize usage by paying users to use the network to meet its intended target. The interpretation is simple: if the base demand of the network is not enough to meet the target usage, then the network has an incentive to subsidize users until the marginal cost of the target usage matches the subsidy amounts. We note that this may only apply to very specific transaction types in practice, as it is difficult to issue subsidies in an incentive-compatible manner that doesn't encourage the inclusion of "junk" transactions.

Maximum price. There often exist prices past which transaction producers would always prefer to not submit a transaction (or, more generally, will only submit transactions that consume no resources, if such transactions exist). In fact, we can characterize the set of all such prices.

To do this, write $S_0 \subseteq S$ for the set of transactions bundles that use no resources, defined

$$S_0 = \{x \in S \mid Ax = 0\}.$$

If $0 \in S$ then S_0 is nonempty (as $0 \in S_0$), and we usually expect this set to be a singleton, $S_0 = \{0\}$. Otherwise, we are saying that there are transactions that are always costless to include. Now, we will define the set

$$P = \bigcap_{x \in S \setminus S_0} \{p \in \mathbb{R}_+^m \mid p^T Ax > q^T x\}.$$

This is the set of prices $p \in P$ such that, for every possible transaction bundle $x \in S$, the price of this transaction bundle, $p^T Ax$, paid to the network, is strictly larger than the total welfare generated by including it, which is $q^T x$. (That is, any transaction bundle x that is not costless is always strictly worse than no transaction, for transaction producers, at these prices.) The set P is nonempty since $\mathbf{1}^T Ax > 0$ for every $x \in S \setminus S_0$ (and $S \setminus S_0$ is finite) so, setting $p = t\mathbf{1}$, we have that

$$p^T Ax - q^T x = t\mathbf{1}^T Ax - q^T x \to \infty > 0,$$

as $t \to \infty$, so $t\mathbf{1} \in P$ for t large enough. The set P is also a convex set, as it is the intersection of convex sets. Additionally, if $p \in P$, then any prices p' satisfying $p' \geq p$ must also have $p' \in P$, where the inequality is taken elementwise. In English: if certain resource prices $p \in P$ would mean that transactions that consume resources are not included, then increasing the price of any resources to $p' \geq p$ also implies the same.

3.3 Solution methods

As mentioned before, the dual problem (12) is convex. This means that it can often be easily solved if the function g (or its subgradients) can be efficiently evaluated. We will see that, assuming users and miners are approximately solving problem (11), we can retrieve approximate (sub)gradients of g and use these to (approximately) solve the dual problem (12). In a less-constrained computational environment, a quasi-Newton method (*e.g.*, L-BFGS) would converge quickly to the optimal prices and be efficient to implement. However, these methods aren't amenable to on-chain computation due to their memory and computational requirements. To solve for the optimal fees on chain, we therefore propose a modified version of gradient descent which is easy to compute and does not require additional storage beyond the fees themselves.

Projected gradient descent. A common algorithm for unconstrained function minimization problems, such as problem (12), is *gradient descent*. In gradient descent, we are given an initial starting point p^0 and, if the function g is differentiable, we iteratively update the prices in the following way:

$$p^{k+1} = p^k - \eta \nabla g(p^k).$$

Here, $\eta > 0$ is some (usually small) positive number referred to as the "step size" or "learning rate" and $k = 0, 1, \dots$ is the iteration number. This rule has a few important properties. For example, if $\nabla g(p^k) = 0$, that is, p^k is optimal, then this rule does not update the prices, $p^{k+1} = p^k$; in other words, any minimizer of g is a fixed point of this update rule. Additionally, this rule can be shown to converge to the optimal value under some mild conditions on g, *cf.* [9, §1.2]. This update also has a simple interpretation: if $\nabla g(p^k)$ is not zero, then a small enough step in the direction of $\nabla g(p^k)$ is guaranteed to evaluate to a lower value than p^k, so an update in this direction decreases the objective g. (This is why the parameter η is usually chosen to be small.)

Note that if the effective domain of the function g, $\mathbf{dom}\, g$, is not $\mathbb{R}^m$, then it is possible that the $(k+1)$st step ends up outside of the effective domain, $p^{k+1} \notin \mathbf{dom}\, g$, so $g(p^k) = \infty$ which would mean that the gradient of g at price p^{k+1} would not exist. To avoid this, we can instead run *projected* gradient descent, where we project the update step into the domain of g, in order to get $p^{k+1} \in \mathbf{dom}\, g$, *i.e.*,

$$p^{k+1} = \mathbf{proj}(p^k - \eta \nabla g(p)) \tag{15}$$

where $\mathbf{proj}(z)$ is defined

$$\mathbf{proj}(z) = \operatorname*{argmin}_{p \in \mathbf{dom}\, g} \|z - p\|_2^2.$$

In English, $\mathbf{proj}(z)$ is the projection of the price to the nearest point in the domain of g, as measured by the sum-of-squares loss $\|\cdot\|_2^2$. (This always exists and is unique as the domain of g is always closed and convex, for any loss function ℓ as defined above.) There is relatively rich literature on the convergence of projected gradient descent, and we refer the reader to [50, 9] for more.

Evaluating the gradient. In general, since we do not know q, we cannot evaluate the function g at a certain point, say p^k. On the other hand, the gradient of g at p^k, when g is differentiable, depends only on the solution to problem (11) and the maximizer for the conjugate function ℓ^*, at the price p^k. (This follows from the gradient equation in (13).) So, if we know that transaction producers are solving their welfare maximization problem (11) to (approximate) optimality, equation (13) suggests a simple descent algorithm for solving the dual problem (12).

From before, let $y^\star$ be a maximizer of $\sup_y \left(y^T p^k - \ell(y)\right)$, which is usually easy to compute in practice, and let x^0 be an (approximate) solution to the transaction inclusion problem (11) (observed, *e.g.*, after the block is built with resource prices p^k). We can approximate the gradient of g at the current fees p using (13), where we replace the true solution $x^\star$ with the observed solution x^0. Since x^0 is Boolean, we can compute the resource usage Ax^0 after observing only the included transactions. We can then update the fees p^k in, say, block k, to a new value p^{k+1} by using projected gradient descent with this new approximation:

$$p^{k+1} = \mathbf{proj}(p^k - \eta(y^\star - Ax^0)). \tag{16}$$

Discussion. Whenever ℓ is differentiable at $y^\star$, we have that $\nabla\ell(y^\star) = p$. (To see this, apply the first-order optimality conditions to the objective in the supremum in the definition of $\ell^\star$.) We can then think of $y^\star$ as the resource utilization such that the marginal cost to the network $\nabla\ell$ is equal to the current fees p. Thus, the network aims to set p such that the realized resource utilization is equal to $y^\star$. We can see that (15) will increase the network fee for a resource being overutilized and decrease the network fee for a resource being underutilized. This pricing mechanism updates fees to disincentivize future users and miners from including transactions that consume currently-overutilized resources in future blocks. Additionally, we note that algorithms of this form are not the only algorithms which are reasonable. For example, any algorithm that has a fixed point p satisfying $\nabla g(p) = 0$ and converges to this point under suitable conditions is also similarly reasonable. One well-known example is an update rule of the form of (3):

$$p^{k+1} = p^k \odot \exp(-\eta \nabla g(p^k)),$$

when the prices must be nonnegative, *i.e.*, when $\mathbf{dom}\, g \subseteq \mathbb{R}^m_+$. We note that one important part of reasonable rules is that they only depend on (an approximation of) the gradient of the function g, since the value of g may not even be known in practice. Additionally, in some cases, the function g is nondifferentiable at prices p. In this case, the subgradient still often exists and convergence of the update rule (16) can be guaranteed under slightly stronger conditions. (The modification is needed as not all subgradients are descent directions.)

Simple examples. We can derive specific update rules by choosing specific loss functions. For example, consider the loss function

$$\ell(y) = \begin{cases} 0 & y = b^\star \\ +\infty & \text{otherwise,} \end{cases}$$

which captures infinite unhappiness of the network designer for any deviation from the target resource usage $b^\star$. The corresponding conjugate function is

$$\ell^*(p) = \sup_y (y^T p - \ell(y)) = (b^\star)^T p,$$

with maximizer $y^\star = b^\star$. (Note that this maximizer does not change for any price p). Since $\mathbf{dom}\, g = \mathbb{R}^m$, then the update rule is

$$p^{k+1} = p^k - \eta(b^\star - Ax^0). \tag{17}$$

If the utilization Ax^0 lies far below $b^\star$, the fees p^k might become negative, *i.e.*, the network would subsidize certain resource usage to meet the requirement that it must be equal to $b^\star$.

Nondecreasing separable loss. A more reasonable family of loss functions would be the nondecreasing, separable losses:

$$\ell(y) = \sum_{i=1}^m \phi_i(y_i).$$

From §3.2 we know that the domain of g is precisely the nonnegative orthant when the functions ϕ_i are superlinear (*i.e.*, satisfy (14)) and bounded from below, so we have that $\mathbf{proj}(z) = (z)_+$, where $(w)_+ = \max\{0, w\}$ for scalar w and is applied elementwise for vectors. Additionally, using the definition of the separable loss, we can write

$$\ell^*(p) = \sum_{i=1}^m \sup_{y_i} \left(y_i p_i - \phi_i(y_i) \right).$$

Letting $y_i^\star$ be the maximizers for the individual problems at the current price p^k, we have

$$p^{k+1} = (p^k - \eta(y^\star - Ax^0))_+.$$

For example, if ϕ_i is an indicator function with $\phi_i(y_i) = 0$ if $y_i \le b_i^\star$ and ∞ otherwise, as in the loss (5), then an optimal point is always $y_i^\star = b_i^\star$, when $p_i \ge 0$. Since no updates will ever set $p_i < 0$, we therefore have,

$$p^{k+1} = (p^k - \eta(b^\star - Ax^0))_+,$$

which is precisely the update given in (2).

Exponential update rule. A log transform of the prices leads to an update rule resembling (3), proposed by Ethereum developers [18]. We assume that the loss function is separable and nondecreasing (*cf.*, §3.2) and that a minimum demand condition is met (given in the full version [27, §3.2]) so that the prices are strictly positive. As a result, we can write the prices p as $p = \exp(\tilde{p})$ for some $\tilde{p} \in \mathbb{R}^m$, where the exponential is applied elementwise. The gradient with respect to $\tilde{p}$ is $\nabla g(\tilde{p}) = \nabla g(p) \odot \exp(\tilde{p})$. Writing the resulting gradient update for $\tilde{p}$ and then taking the exponential of both sides, we get

$$p^{k+1} = p^k \odot \exp\left(-p^k \odot \eta \nabla g(p^k)\right).$$

When we use the indicator loss function (4), we obtain a rule similar to the original EIP-1559 update (3) but with an extra factor of p^k in the exponent:

$$p^{k+1} = p^k \odot \exp(\eta(p^k \odot (Ax^k - b^\star))). \tag{18}$$

The lack of this extra factor in EIP-1559 may explain some of the behavior explored in [40].

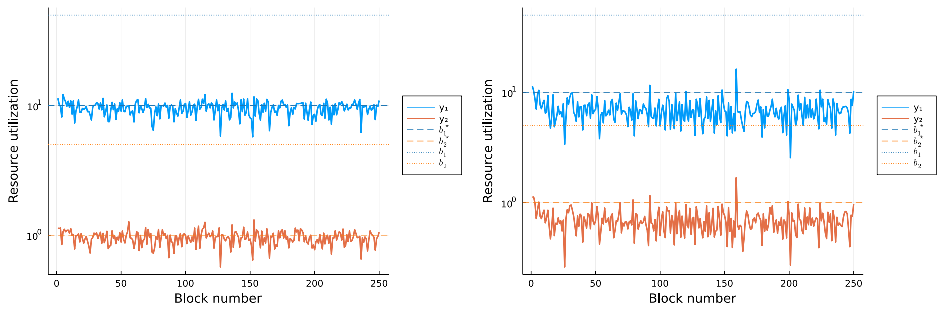


Figure 1 Resource utilization for multidimensional pricing (left) clusters closer to target values than for uniform pricing (right), which includes limited information about individual targets or caps.

4 The cost of uniform prices

In this section, we show that pricing resources using the method outlined above can increase network efficiency and make the network more robust to DoS attacks and resource demand distribution shifts. We construct a toy experiment to illustrate these differences between uniform and multidimensional resource pricing but leave more extensive numerical studies to future work.

The setup. We consider a blockchain system with two resources (*e.g.*, compute and storage) with resource utilizations y_1 and y_2. Resource 1 is much cheaper for the network to use than resource 2, so $b^\star = (10, 1)$ and $y \leq b = (50, 5)$. Furthermore, we assume that there is a joint capacity constraint on these resources, $y_1 + 10y_2 \leq 50$, which captures the resource tradeoff. Each transaction a_j is therefore a vector in $\mathbb{R}^3_+$ with $a_j = (a_{1j}, a_{2j}, a_{1j} + 10a_{2j})$. As in §3.3, we consider the simple loss function $\ell(y) = 0$ if $y = b^\star$ and $+\infty$ otherwise, which has the update rule given in (17). In the scenarios below, we compare our multidimensional fee market approach to a baseline, where both resources are combined into one equal to $a_{1j} + 10a_{2j}$ with $b^\star = 20\% \times \max(b_1, b_2) = 10$. We demonstrate that pricing these resources separately leads to better network performance. All code is available at

`https://github.com/bcc-research/resource-pricing`.

We run simulations in the Julia programming language [10]. The transaction producers' optimization problem (11) is modeled with JuMP [28] and solved with COIN-OR's simplex-based linear programming solver, Clp [32]. The solution is usually integral, but when it is not, we fall back to the HiGHS mixed-integer linear program solver [35].

Scenario 1: steady state behavior. We consider a sequence of 250 blocks. At each block, there are 15 submitted transactions, with resource usage randomly drawn as $a_{1j} \sim \mathcal{U}(0.5, 1)$ and $a_{2j} \sim \mathcal{U}(0.05, 0.1)$. (For example, these may be moderate compute and low storage transactions.) Transaction utility is drawn as $q_j \sim \mathcal{U}(0, 5)$. We initialize the price vector as $p = 0$ and examine the steady state behavior, where the price updates and transaction producer behavior are defined as in the previous section. We use a learning rate $\eta = 1 \times 10^{-2}$ throughout. The resource utilization, shown in figure 1, suggests that our multidimensional scheme more closely tracks the target utilities $b^\star$ than a single-dimensional fee market. Furthermore, the number of transactions included per block is consistently higher, illustrated in figure 2 (purple line).

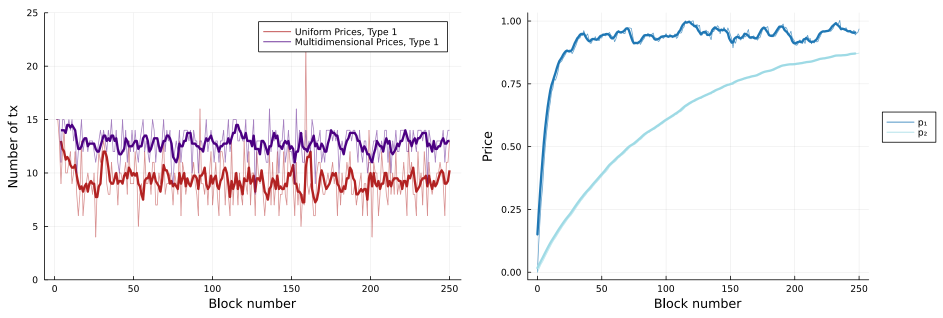


Figure 2 Multidimensional pricing allows us to include more transactions per block (left) by optimally adjusting prices (right). The thicker line is the four-sample moving average of the data.

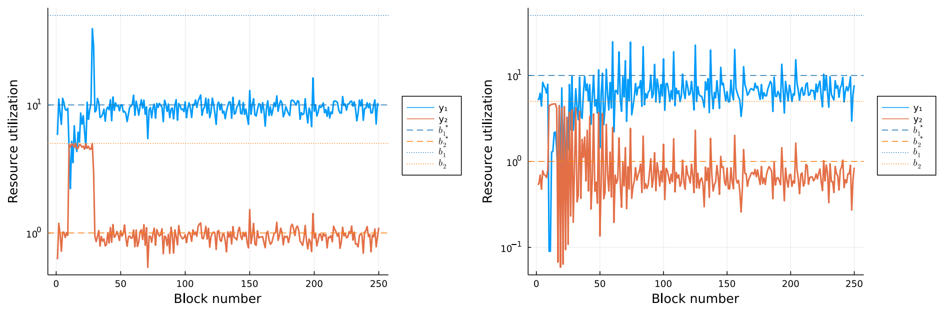


Figure 3 Resource utilization for multidimensional pricing (left) clusters closer to target values than for uniform pricing (right) after a burst to the resource limit to handle transactions that make heavy use of resource 2.

Scenario 2: transactions distribution shift. Often, the distribution of transaction types submitted to a blockchain network differs for a short period of time (*e.g.*, during NFT mints). There may be a change in both the number of transactions and the distribution of resources required. We repeat the above simulation but add 150 transactions in block 10 with resource vector $a_j = (0.01, 0.5)$. (For example, these transactions may have low computation but high storage requirements.) We draw the utility $q_j \sim \mathcal{U}(10, 20)$ and begin the network at the steady-state prices from scenario 1. In figure 3, we see that a multidimensional fee market gracefully handles the distribution shift. The network fully utilizes resource 2 for a short period of time before returning to steady state. Uniform pricing, on the other hand, does not do a good job of adjusting its resource usage and oscillates around the target. Figure 4 show that, as a result, multidimensional pricing is able to include more transactions, both during the distribution shift and after the network returns to steady state. We see that the prices smoothly adjust accordingly.

5 Extensions

Parallel transaction execution model. Consider the scenario where the nodes have L parallel execution environments (*e.g.*, threads), each of which has its own set of m identical resources. In addition, there are r resources shared between the environments. We denote transactions run on thread k by $x_k \in \{0,1\}^n$. The resource allocation problem becomes

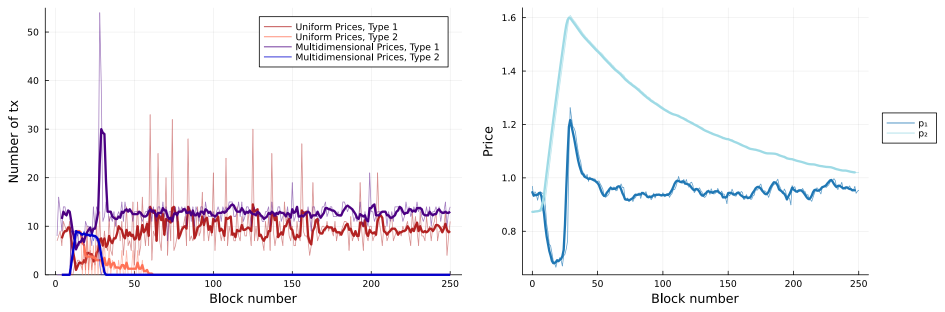


Figure 4 Multidimensional pricing allows us to include more transactions per block by optimally adjusting prices (right). The thicker line is the four-sample moving average of the data.

$$
\begin{array}{ll}
\text{maximize} & \sum_{k=1}^{L} q^T x_k - \ell(y_1, \dots, y_L, y^{\text{shared}}) \\
\text{subject to} & y_k = Ax_k, \quad x_k \in \mathbf{conv}(S) \qquad k = 1, \dots, L \\
& y^{\text{shared}} = Bz, \quad z = \sum_{k=1}^{L} x_k, \quad z \in \mathbf{conv}(S^{\text{shared}})
\end{array}
$$

As before, the Boolean vector sets S and S^{shared} encode constraints such as resource limits for each parallel environment and the shared environment respectively. In addition, we'd expect to have $z \leq \mathbf{1}$ if each transaction is only allocated to a single environment, which can be encoded by S^{shared}. By stacking the variables into one vector, this problem can be seen as a special case of (8) and can be solved with the same method presented in this work. (The interpretation here is that we are declaring a number of "combined resources", each corresponding to the parallel execution environments along with their shared resources.)

Different price update speeds. Some resources may be able to sustain burst capacities for much shorter periods of times than other resources. In practice, we may wish to increase the prices of these resources faster. For example, a storage opcode that generates a lot of memory allocations will quickly cause garbage collection overhead, which could slow down the network. We can update (15) to include a learning rate for each resource. These learning rates can be chosen by system designers using simulations and historical data. We collect these in a diagonal matrix $D = \mathbf{diag}\,(\eta_1, \dots, \eta_m)$ and change the update rule to be

$$
p^{k+1} = \mathbf{proj}(p^k - D\nabla g(p^k)).
$$

Contract throughput. Alternatively, we can define utilization on a per-contract basis instead of a per-resource basis (per-contract fees were recently proposed by the developers of Solana [54]). We define the utilization of a smart contract j as $z_j = (w^T a_j)x_j$, where w is some weight vector and $x_j \in \mathbb{Z}_+$ is the number of times contract j is called. In matrix form, $z = A^T w \odot x$, where $\odot$ is the Hadamard (elementwise) product. For each contract, the utilization z_j is 0 when $x_j = 0$, which can be interpreted as not calling contract j in a block. When $x_j > 0$, the utilization is $\left(\sum_i w_i a_{ij}\right) x_j$. When we use per-contract utilizations, the loss function can capture a notion of fairness in resource allocation to contracts. For example, we may want to prioritize cheaper-to-execute contracts over more expensive ones by using, *e.g.*, proportional fairness as in [36], though there are many other notions that may be useful. With this setup, the resource allocation problem is

$$\begin{array}{ll} \text{maximize} & q^T x - \ell(z) \\ \text{subject to} & z = A^T w \odot x \\ & x \in \mathbf{conv}(S). \end{array}$$

Again, we can introduce the dual variable $p \in \mathbb{R}^n$ for the equality constraint, and, with a similar method to the one introduced in this paper, iteratively update this variable to find the optimal fees to charge for each smart contract call.

6 Conclusion

We constructed a framework for multidimensional resource pricing in blockchains. Using this framework, we modeled the network designer's goal of maximizing transaction producer utility, minus the loss incurred by the network, as a an optimization problem. We used tools from convex optimization – and, in particular, duality theory – to decompose this problem into two simpler problems: one solved on chain by the network, and another solved off chain by the transaction producers. The prices that unify the competing objectives of minimizing network loss and maximizing transaction producer utility are precisely the dual variables in the optimization problem. Setting these prices correctly (*i.e.*, to minimize the dual function) results in a solution to the original problem. We then demonstrated efficient methods for updating prices that are amenable to on-chain computation and derived an EIP-1559-like mechanism as an example. In a simple numerical example, we find that the proposed mechanism allows the network to equilibrate to its resource utilization target more quickly than uniform pricing, while offering greater throughput without increasing node hardware requirements. Finally, we show a number of simple extensions to our framework that capture other proposed mechanisms such as per-contract fees. We find that it allows the network to equilibrate to its resource utilization target more quickly than the uniform price case and offers greater throughput without increasing node requirements.

To the best of the authors' knowledge, this is the first work to systematically study optimal pricing of resources in blockchains in the many-asset setting. Future work and improvements to this model include a detailed game-theoretic analysis, extending that of [31], along with a more concrete analysis of the dynamical behavior of fees set in this manner. Finally, a thorough numerical evaluation of these methods under realistic conditions (such as testnets) will be necessary to see if these methods are feasible in production.

References

1 John Adler. Eip-2242: Transaction postdata, 2019. URL: `https://eips.ethereum.org/EIPS/eip-2242`.

2 John Adler. Multi-threaded data availability on eth 1. Ethresearch, 2019. URL: `https://ethresear.ch/t/multi-threaded-data-availability-on-eth-1/5899`.

3 John Adler. Accounts, strict access lists, and UTXOs - research / execution, 2020. URL: `https://forum.celestia.org/t/accounts-strict-access-lists-and-utxos/37`.

4 John Adler. Wait, it's all resource pricing? EthCC, 2021. URL: `https://www.youtube.com/watch?v=YoWMLoeQGeI`.

5 John Adler. Always has been (or, wait, it's all resource pricing? part 2). EthCC, 2022. URL: `https://www.youtube.com/watch?v=Zq8uwpX39oI`.

6 Akshay Agrawal, Stephen Boyd, Deepak Narayanan, Fiodar Kazhamiaka, and Matei Zaharia. Allocation of fungible resources via a fast, scalable price discovery method. *Mathematical Programming Computation*, pages 1–30, 2022.

7 Mustafa Al-Bassam. LazyLedger: A distributed data availability ledger with client-side smart contracts. *CoRR*, abs/1905.09274, 2019. `arXiv:1905.09274`.

8 Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities, 2018. doi:10.48550/ARXIV.1809.09044.

9 Dimitri P Bertsekas. *Nonlinear Programming*. Athena Scientific, 3 edition, 1999.

10 Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

11 Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

12 Vitalik Buterin. Eip 150: Gas cost changes for io-heavy operations, 2016. URL: https://eips.ethereum.org/EIPS/eip-150.

13 Vitalik Buterin. Geth nodes under attack again, 2016. URL: https://www.reddit.com/r/ethereum/comments/55s085/geth_nodes_under_attack_again_we_are_actively/?st=itxh568s&sh=ee3628ea.

14 Vitalik Buterin. Transaction spam attack: Next steps, 2016. URL: https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/.

15 Vitalik Buterin. Easy parallelizability issue #648 ethereum/EIPs, 2017. URL: https://github.com/ethereum/EIPs/issues/648.

16 Vitalik Buterin. First and second-price auctions and improved transaction-fee markets, 2018. URL: https://ethresear.ch/t/first-and-second-price-auctions-and-improved-transaction-fee-markets/2410.

17 Vitalik Buterin. An incomplete guide to rollups, 2021. URL: https://vitalik.ca/general/2021/01/05/rollup.html.

18 Vitalik Buterin. Multidimensional eip 1559. Ethresearch, 2022. URL: https://ethresear.ch/t/multidimensional-eip-1559/11651.

19 Vitalik Buterin. Proto-danksharding FAQ, 2022. URL: https://notes.ethereum.org/@vbuterin/proto_danksharding_faq.

20 Vitalik Buterin. State of research: Increasing censorship resistance of transactions under proposer/builder separation (pbs), 2022. URL: https://notes.ethereum.org/@vbuterin/pbs_censorship_resistance.

21 Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. Eip-1559: Fee market change for eth 1.0 chain, 2019. URL: https://eips.ethereum.org/EIPS/eip-1559.

22 Vitalik Buterin and Martin Swende. Eip-2929: Gas cost increases for state access opcodes, 2020. URL: https://eips.ethereum.org/EIPS/eip-2929.

23 Vitalik Buterin and Martin Swende. EIP-2930: Optional access lists, 2020. URL: https://eips.ethereum.org/EIPS/eip-2930.

24 Yang Chen, Zhongxin Guo, Runhuai Li, Shuo Chen, Lidong Zhou, Yajin Zhou, and Xian Zhang. Forerunner: Constraint-based speculative transaction execution for ethereum (full version). In *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*. ACM, 2021. doi:10.1145/3477132.3483564.

25 Hao Chung and Elaine Shi. Foundations of transaction fee mechanism design. *arXiv preprint arXiv:2111.03151*, 2021.

26 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

27 Theo Diamandis, Alex Evans, Tarun Chitra, and Guillermo Angeris. Dynamic pricing for non-fungible resources: Designing multidimensional blockchain fee markets. *arXiv preprint arXiv:2208.07919*, 2022.

28 Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM review*, 59(2):295–320, 2017.

29 Ethereum. Gas and fees, 2022. URL: https://ethereum.org/en/developers/docs/gas/.

30 Ethereum. Run a node, 2022. URL: `https://ethereum.org/en/run-a-node/`.

31 Matheus Ferreira, Daniel Moroz, David Parkes, and Mitchell Stern. Dynamic posted-price mechanisms for the blockchain transaction-fee market. In *Proceedings of the 3rd ACM conference on Advances in Financial Technologies*, pages 86–99, 2021.

32 John Forrest, Stefan Vigerske, Ted Ralphs, Lou Hafer, John Forrest, jpfasano, Haroldo Gambini Santos, Matthew Saltzman, Jan-Willem, Bjarni Kristjansson, h-i gassmann, Alan King, pobonomo, Samuel Brito, and to st. coin-or/clp: Release releases/1.17.7, January 2022. `doi:10.5281/zenodo.5839302`.

33 Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Yu Xia, Runtian Zhou, and Dahlia Malkhi. Block-STM: Scaling blockchain execution by turning ordering curse to a performance blessing, 2022. `doi:10.48550/ARXIV.2203.06871`.

34 Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

35 Qi Huangfu and JA Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

36 Frank Kelly. Charging and rate control for elastic traffic. *European transactions on Telecommunications*, 8(1):33–37, 1997.

37 Kshitij Kulkarni, Theo Diamandis, and Tarun Chitra. Towards a theory of maximal extractable value i: Constant function market makers, 2022. `arXiv:2207.11835`.

38 Fuel Labs. GitHub - FuelLabs/fuel-specs: Specifications for the fuel protocol and the FuelVM, a blazingly fast blockchain VM, 2022. URL: `https://github.com/FuelLabs/fuel-specs`.

39 Stefanos Leonardos, Barnabé Monnot, Daniël Reijsbergen, Efstratios Skoulakis, and Georgios Piliouras. Dynamical analysis of the eip-1559 ethereum fee market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 114–126, 2021.

40 Stefanos Leonardos, Daniël Reijsbergen, Daniël Reijsbergen, Barnabé Monnot, and Georgios Piliouras. Optimality despite chaos in fee markets. *arXiv preprint arXiv:2212.07175*, 2022.

41 Steven Low. A duality model of tcp and queue management algorithms. *IEEE/ACM Transactions On Networking*, 11(4):525–536, 2003.

42 Steven Low and David Lapsley. Optimization flow control. i. basic algorithm and convergence. *IEEE/ACM Transactions on networking*, 7(6):861–874, 1999.

43 Kamilla Nazirkhanova, Joachim Neu, and David Tse. Information dispersal with provable retrievability for rollups, 2021. `doi:10.48550/ARXIV.2111.12323`.

44 Daniel Perez and Benjamin Livshits. Broken metre: Attacking resource metering in evm. *arXiv preprint arXiv:1909.07220*, 2019.

45 Polygon Team. Introducing avail by polygon, a robust general-purpose scalable data availability layer, 2021. URL: `https://blog.polygon.technology/introducing-avail-by-polygon-a-robust-general-purpose-scalable-data-availability-layer/`.

46 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? *arXiv preprint arXiv:2101.05511*, 2021.

47 Daniël Reijsbergen, Shyam Sridhar, Barnabé Monnot, Stefanos Leonardos, Stratis Skoulakis, and Georgios Piliouras. Transaction fees on a honeymoon: Ethereum's eip-1559 one month later. In *2021 IEEE International Conference on Blockchain (Blockchain)*, pages 196–204. IEEE, 2021.

48 Tim Roughgarden. Transaction fee mechanism design. *SIGecom Exch.*, 19(1):52–55, 2021. `doi:10.1145/3476436.3476445`.

49 Vikram Saraph and Maurice Herlihy. An empirical study of speculative concurrency in ethereum smart contracts, 2019. `doi:10.48550/ARXIV.1901.01376`.

50 Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*, volume 3. Springer Series in Computational Mathematics, 1985.

51 Ertem Nusret Tas, Dionysis Zindros, Lei Yang, and David Tse. Light clients for lazy blockchains, 2022. _eprint: 2203.15968.

52 Jeffrey Wilcke. The ethereum network is currently undergoing a DoS attack, 2016. URL: `https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/`.

53 Anatoly Yakovenko. Sealevel, parallel processing thousands of smart contracts, 2020. URL: `https://medium.com/solana-labs/sealevel-parallel-processing-thousands-of-smart-contracts-d814b378192`.

54 Anatoly Yakovenko. Consider increasing fees for writable accounts issue #21883 solana-labs/solana, 2021. URL: `https://github.com/solana-labs/solana/issues/21883`.

Security Analysis of Filecoin's Expected Consensus in the Byzantine vs Honest Model

Xuechao Wang[1]
Thrust of Financial Technology, HKUST(GZ), Guangzhou, China

Sarah Azouvi
Protocol Labs, San Francisco, CA, USA

Marko Vukolić
Protocol Labs, San Francisco, CA, USA

Abstract

Filecoin is the largest storage-based open-source blockchain, both by storage capacity (>11EiB) and market capitalization. This paper provides the first formal security analysis of Filecoin's consensus (ordering) protocol, Expected Consensus (EC). Specifically, we show that EC is secure against an arbitrary adversary that controls a fraction β of the total storage for $\beta m < 1 - e^{-(1-\beta)m}$, where m is a parameter that corresponds to the expected number of blocks per round, currently $m = 5$ in Filecoin. We then present an attack, the n-split attack, where an adversary splits the honest miners between multiple chains, and show that it is successful for $\beta m \geq 1 - e^{-(1-\beta)m}$, thus proving that $\beta m = 1 - e^{-(1-\beta)m}$ is the tight security threshold of EC. This corresponds roughly to an adversary with 20% of the total storage pledged to the chain. Finally, we propose two improvements to EC security that would increase this threshold. One of these two fixes is being implemented as a Filecoin Improvement Proposal (FIP).

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Decentralized storage, Consensus, Security analysis

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.5

Related Version *Full Version*: `https://arxiv.org/pdf/2308.06955.pdf` [29]

Acknowledgements The authors would like to thank Guy Goren for his suggestion of Consistent Broadcast as a mitigation to the n-split attack.

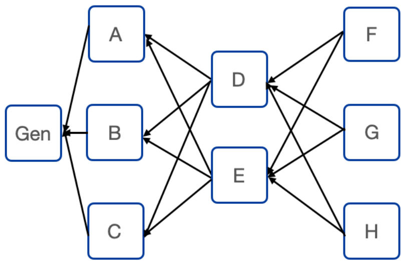
1 Introduction

Filecoin is the largest storage-based blockchain in terms of both market cap [3] and total raw-byte storage capacity (>11EiB) [6]. In Filecoin, miners, called Storage Providers (SPs), gain the right to participate in the consensus protocol and to create blocks by pledging storage capacity to the chain.[2] They are in return compensated with a financial reward in the form of newly minted FIL, the cryptocurrency underlying Filecoin, whenever their blocks are included on-chain, where probability of a miner minting new block corresponds to their storage power. The Filecoin consensus mechanism Storage Power Consensus (SPC) consists mainly of two components: first, a *Sybil-resistance mechanism* that keeps an accurate map of the storage pledged by each storage provider; and second, a consensus protocol that can be run by any set of weighted participants and outputs an ordered list of transactions.

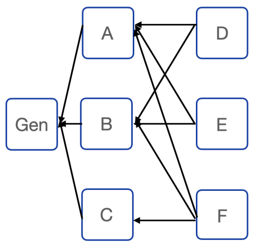
[1] Corresponding author, part of work was done at Protocol Labs.
[2] Filecoin further incentivizes the storage of "useful" data, where SPs have the additional opportunity to boost their raw-byte storage power, offering deals to verified clients, yielding *quality adjusted* power [2].

In this paper, we ignore the mechanisms that keep the mapping between miners and their respective storage accurate (i.e., the Sybil-resistance mechanism and the quality adjusted power policy) and focus on the sub-protocol run by the weighted miners to produce an ordered list of transactions. This sub-protocol is called *Expected Consensus* (EC) and weighs each participant according to their storage power. We assume that the weighted list of miners is accurately maintained and given as an input to EC. EC is a longest-chain style protocol [1] (or, more accurately, a heaviest-chain protocol). At a high level, it operates by running a leader election at every time slot in which, on expectation, m participants may be eligible to submit a block, where m is a parameter currently equal to 5. Each participant is elected with probability proportional to their weight. Multiple valid blocks submitted in a given round may form a *tipset*, which is a set of blocks sharing the same height (i.e., round number) and parent tipset. In EC, the blockchain is a chain of tipsets (i.e., a directed acyclic graph [DAG] of blocks) rather than a chain of blocks. For example, in Figure 1a blocks {A,B,C}, {D,E} and {F,G,H} each form a different tipset; and in Figure 1b blocks {A,B,C}, {D,E} and {F} each form a different tipset. Every block in a tipset adds weight to its chain of tipsets, while the fork choice rule is to choose the heaviest tipset. EC works in a very similar fashion as longest-chain protocols like Bitcoin do, but it uses tipsets instead of blocks. EC's security has, until now, only been argued informally, as with Bitcoin in its early days. Intuitively, tipsets make it harder for an adversary with less storage to form a competing chain of tipsets with more blocks than the main chain, as miners can create a number of blocks proportional to their storage power. This is analogous to Nakamoto's private attack on the longest-chain protocol [22]. Specifically, assuming that two competing chains of tipsets are growing, with different amount of storage power producing the two, since in EC more than one block can be appended to a chain at each round, the difference between the number of blocks created on each chain will grow roughly m times faster than in the case without tipset (i.e., where each chain can grow by at most one block at each round). However, this intuitive security justification only applies when examining the private attack, a specific instance, not a general adversary. The lessons learned from the balance attack [23] on GHOST [28] highlight that a comprehensive analysis encompassing all potential attacks is crucial for assuring the security of a blockchain protocol. This comprehensive evaluation is the main focus of this paper.

In this work, we conduct a formal security analysis of EC and prove that EC is secure against any adversary that owns a fraction β of the total storage power for $\beta m < 1 - e^{-(1-\beta)m}$ (Section 5). To achieve this, we carefully extend the proof technique developed in [14] to EC: the key step is to identify the sufficient condition for a block to stay in the chain forever, regardless of the complex DAG structure in EC. Indeed, the incorporation of tipsets introduces substantial complexities to the problem. For instance, in the longest-chain protocol, the chain growth is an independent and identically distributed random variable in each round. Conversely, within EC, the chain growth becomes dependent on the entire history of the blockchain, given that it depends on the structure of the DAG. This increased dependency adds layers of intricacy to the security analysis. Following similar literature [14, 16, 19], we consider a rather strong adversary, which we specify in Section 2, that has "full control" over both the network and the tie breaking rule. We then propose an attack, the n-split attack, in Section 6 in which an adversary with power β such that $\beta m \geq 1 - e^{-(1-\beta)m}$ can break the security of EC, effectively proving that the security threshold $\beta m = 1 - e^{-(1-\beta)m}$ is tight for EC. With current parameters, this means that EC is secure against an adversary that holds roughly 20% of the total storage power. In our attack, an elected leader, controlled by the adversary, equivocates by sending different blocks to different miners at each round with the aim to split the honest miners into different chains and thus reducing the weight



(a) Example of a tipset chain: {A,B,C} is the first tipset after Genesis, and the parent tipset of {D,E}, which is itself the parent tipset of tipset {F,G,H}.



(b) Example of two tipsets with equal weight. Tipset {D,E} and tipset {F} both have a weight of 5. (roughly, the weight is calculated as the sum of the blocks in all the tipsets, see Section 3.3 for details).

Figure 1 Tipset structure in EC.

of each tipsets' chain. While the honest participants are split and all mine on potentially many different forks, the adversary can construct a *private* tipsets' chain on the side, i.e., a chain that does not include any block mined by an honest miner (called for simplicity *honest block*) and that will not be broadcast to any honest miner until the end of the attack. Finally, in Section 7, we propose two countermeasures against this attack aimed at augmenting the security threshold of Filecoin. The first one entails eliminating the concept of tipsets, substituting EC with the longest-chain protocol [19, 13, 9], as our observations suggest that lowering the value of m can enhance the security threshold. The second approach involves the adoption of a consistent or reliable broadcast protocol [12, 18] to prohibit the adversary from equivocating. Our second countermeasure is currently in the process of being implemented as a Filecoin Improvement Proposal [7].

Related Work

This work is directly inspired by the line of work formally analyzing longest-chain protocols either in the proof-of-work case [16, 27, 20] or proof-of-stake [10, 19, 14, 25]. We adapt the technique in [14] to account for tipsets instead of blocks (see Lemma 5). The main difference between using a chain of tipsets and a chain of blocks is that in the tipset case, the number of blocks in the chain at each round can increase by any finite integer value and also depends on the structure of the DAG. By contrast, in the chain-of-blocks case, the number of blocks of any honest chain increases by zero or one at each round. This makes the tipset analysis more complex.

Similar attacks to the one we describe in Section 6 were proposed by Bagaria et al. [10] in the context of proof-of-stake and by Natoli and Gramoli [23] in the proof-of-work context. In both these works, the attacks are described on a DAG-based blockchain, where each new block can include any previous block as its parent. We adapt it to the tipset case, which is slightly different: in the case of tipsets, a block may have multiple parents, but only blocks that have themselves the same set of parents can be referenced by the child block. The idea behind these attacks is to have an adversary publish its blocks in a timely manner on different forks to ensure that honest miners keep on extending two or even more chains instead of one, effectively spreading their *power* (be it stake, computation or storage) on different chains.

2 Model

In this section we present our model and assumptions.

2.1 Participants

Filecoin requires participants to pledge storage capacity to the chain to be added to the list of participants. Following the work in [16, 19] we consider a flat model, meaning that each participant accounts for one unit of storage. This could easily be extended to a non-flat model by considering that a participant holding x units of storage controls x "flat" participants. We consider a static model wherein the set of participants is fixed during the execution of the protocol. We assume that each participant i possesses a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ and that every participant is aware of the other participants and their respective public keys.

We consider a static adversary that corrupts a fraction β of the participants at the beginning of the execution of the protocol. In order to defend against an adaptive adversary who can corrupt honest nodes on the fly (i.e., dynamically, at the start of each round), one can either use key evolving signature schemes [13] or checkpointing [8]. However, in order to keep the problem simple, we do not consider an adaptive adversary in this paper.

2.2 Network assumptions

We consider the lock-step synchronous network model adopted in [16, 19]. Time is divided into synchronized rounds, each indexed with an integer in $\mathbb{N}$. Following Filecoin's terminology [1], we refer to each time slot as an *epoch*. Each epoch has a fixed duration (currently 30 seconds in Filecoin). To abstract the underlying peer-to-peer gossip network in Filecoin, we simply assume that all messages sent by honest nodes are broadcast to all nodes and that all honest nodes re-broadcast any message they deliver. All network messages are delivered by the adversary, and we allow the adversary to selectively delay messages sent by honest nodes, with the following restrictions: (i) the messages sent in an epoch must be delivered by the end of the current epoch; and (ii) the adversary cannot forge or alter any message sent by an honest node. The adversary does not suffer any network delay. Note that the adversary can selectively send its message only to a subset of honest nodes. However, due to the re-broadcast mechanism, all honest nodes will receive the message by the end of the next epoch.

The non-lock-step synchronous model, also known as the Δ-synchronous model, is also frequently employed in blockchain security analysis [9, 13, 14, 24, 27]. This model ensures messages sent by honest nodes are delivered within a sliding window of Δ epochs. While this model might be more suitable for proof-of-work blockchains, where miners persistently mine and broadcast blocks, its application becomes less pertinent for PoS blockchains. In the latter, honest nodes primarily remain dormant, awaiting the epoch boundaries to send messages. Given the efficiency of today's network infrastructure, a 30-second window adopted by EC is quite conservative. Consequently, we find little justification to incorporate the Δ-synchronous model in our EC analysis.

2.3 Randomness

Random beacon. A random beacon [26] is a system that emits a random number at regular intervals. EC relies on drand [4], a decentralized random beacon, to provide miners a different random number at each epoch. This service is run by a set of 16 independent institutions that run a multi-party protocol to output, at regular intervals, a fresh random number. We assume that this random number is unbiasable (i.e., truly random) and unpredictable before the beginning of the epoch. We also assume that each miner in Filecoin has the same view of each drand output, i.e., that drand is secure. We denote drand_i the random beacon emitted by drand and used by Filecoin miners at epoch i.

Verifiable Random Function. A Verifiable Random Function (VRF) [21] is a function that outputs a random number in a verifiable way, i.e., everyone can verify that the output is indeed random and was generated correctly. A VRF is composed of two polynomial-time algorithms: $\mathsf{VRF.Proof}$ and $\mathsf{VRF.Verify}$ (we omit the key generation). $\mathsf{VRF.Proof}$ takes as inputs a seed seed and a secret key sk and outputs a tuple $(y = G_{\mathsf{sk}}(\mathsf{seed}), p = \pi_{\mathsf{sk}}(\mathsf{seed}))$ where y is a random number and p is a proof that can be used to verify the correctness of y. $\mathsf{VRF.Verify}$ takes as input a tuple (seed, y, p) and a public key pk and uses p to verify that $y = G_{\mathsf{sk}}(\mathsf{seed})$, in which case it outputs 1; otherwise, it outputs 0. A VRF is correct if:

1. if $(y, p) = \mathsf{VRF.Proof}_{\mathsf{sk}}(\mathsf{seed})$, then $\mathsf{VRF.Verify}_{\mathsf{pk}}(\mathsf{seed}, y, p) = 1$;
2. for all $(\mathsf{sk}, \mathsf{seed})$, there is a unique y s.t. $\mathsf{VRF.Verify}_{\mathsf{pk}}(\mathsf{seed}, y, \pi_{\mathsf{sk}}(\mathsf{seed})) = 1$;
3. $G_{\mathsf{sk}}(\mathsf{seed})$ is computationally indistinguishable from a random number for any probabilistic polynomial-time adversary.

Throughout the rest of this paper, we assume the existence of a correct VRF.

3 Filecoin's Expected Consensus (EC)

Filecoin's consensus protocol, EC, consists of three main components: a leader election sub-protocol, a mining algorithm and a fork choice rule. Briefly, at the beginning of each epoch, participants will check their eligibility to produce a block by running the leader election. If they are elected, they use the fork choice rule to select a tipset and include it as their *parent* before broadcasting their block. We define the protocol more formally in this section. However, we intentionally omit some details, such as those regarding how participants must continually post proofs related to their pledged storage, as they are not relevant to our analysis. Instead, we assume that all participants continuously maintain one unit of storage. Furthermore, in practice in Filecoin [1] a participant with power x that is elected twice in the same epoch will create only one block that *weighs* twice more. This is not relevant to our analysis, so we ignore it and prefer a flat model wherein a participant elected twice simply creates two blocks under two different identities. Such a model will favor an adversary as we illustrate in Section 6 and hence renders our analysis stronger.

Due to space limitations, a pseudocode representation of the algorithms described in this section can be found in Appendix A.

3.1 Leader Selection Protocol

EC's leader election is inspired by Algorand's cryptographic sortition [17]. Briefly, the leader selection relies on a Verifiable Random Function (VRF) [21] that takes as input the drand output value for that epoch. In each epoch, each participant will compute $\mathsf{VRF.Proof}_{\mathsf{sk}}(\mathsf{seed}) = (y = G_{\mathsf{sk}}(\mathsf{seed}), p = \pi_{\mathsf{sk}}(\mathsf{seed}))$ where seed is the drand value. If y is below a predefined value target that is a parameter of the protocol, then that participant is elected leader. Any other participant can then use p in order to verify that the random value y was computed correctly (i.e., $\mathsf{VRF.Verify}_{\mathsf{pk}}(\mathsf{seed}, y, p) = 1$) and that the participant is indeed an elected leader. The value of target is chosen such that on expectation m leaders are elected in each epoch. m is a parameter of the EC protocol currently set to $m = 5$.

Proving that the leader selection mechanism is secure is outside the scope of this paper, as similar results were already proven in, e.g., Algorand [17]. Instead, we assume that in each epoch, there is an independent random number of participants that are elected leaders and that the number of leaders in each epoch follows a Poisson distribution of parameter m. For a coalition that consists of a fraction α of all the participants, their number of elected leaders in an epoch follows a Poisson distribution of parameter $\alpha \times m$.

3.2 Block and Tipset Structure

A block is composed of a header and a payload. The payload includes transactions as well as other messages necessary for maintaining the set of participants up to date. We omit its content in this analysis.

When a participant is elected to create a block, they include in the *header* of the block their proof of eligibility (i.e., the VRF proof), an epoch number (the epoch at which the block was created), a proof of storage called WinningPost to prove that they maintain the storage they have pledged (we omit the details of such proof) and finally a pointer to a set of *parent* blocks. For a block $\mathcal{B}$, we denote $\mathcal{B}.\mathsf{parent}$ its parents set and $\mathcal{B}.\mathsf{epoch}$ its epoch number. The parents of a block must satisfy certain conditions. First, they must all be in the same epoch, and that epoch needs to be smaller than the block's epoch. Second, all parent blocks need to have the same set of parents themselves. Each set of blocks that are in the same epoch and have the same set of parents is called a *tipset* and denoted $\mathcal{T}$. Formally, a tipset $\mathcal{T}$ is a non-empty set of blocks: $\mathcal{T} = \{\mathcal{B}_1, \cdots, \mathcal{B}_r\}$, each of which belongs to the same epoch, i.e., $\mathcal{B}_1.\mathsf{epoch} = \cdots = \mathcal{B}_r.\mathsf{epoch}$ and has the same set of parents, i.e., $\forall(\mathcal{B}_i, \mathcal{B}_j) \in \mathcal{T}^2 : \ \mathcal{B}_i.\mathsf{parent} = \mathcal{B}_j.\mathsf{parent}$. Since all blocks in a tipset have the same parent, we abuse the notation and denote $\mathcal{T}.\mathsf{parent}$ to denote the parent of tipset $\mathcal{T}$. Similarly, $\mathcal{T}.\mathsf{epoch}$ denotes the tipset epoch. We note that $\mathcal{T}.\mathsf{parent}$ is a tipset itself.

Since each block references a set of blocks, a Directed Acyclic Graph (DAG) structure can be inferred from each block or tipset, where the blocks are the vertices and the references to parents are the edges. Similarly, the set of tipsets referencing each other as parents form a *chain*. For example, Figure 1a represents a chain of 4 tipsets (including the genesis) and a blockDAG of 9 blocks. Formally, a chain $\mathcal{C}$ is then a set of ordered tipsets $\mathcal{C} = \{\mathcal{T}_0, \mathcal{T}_1, \cdots, \mathcal{T}_l\}$ such that $\mathcal{T}_i.\mathsf{parent} = \mathcal{T}_{i-1}$ for all $i > 1$. By convention, we have $\mathcal{T}_1.\mathsf{parent} = \mathcal{T}_0 = \{\text{Genesis block}\}$ and $\mathcal{T}_i = \emptyset$ if there is no block in epoch i. We note $\mathcal{C}[\mathcal{T}_i] = \{\mathcal{T}_0, \mathcal{T}_1, \cdots, \mathcal{T}_i\}$. Similarly, for a tipset $\mathcal{T}$, we can infer the associated chain, denoted $\mathcal{C}[\mathcal{T}]$ as follows: $\mathcal{C}[\mathcal{T}] = \{\mathcal{T}_0, \cdots, (\mathcal{T}.\mathsf{parent}).\mathsf{parent}, \mathcal{T}.\mathsf{parent}, \mathcal{T}\}$.

3.3 Fork Choice Rule and Weight Function

In order to decide which tipset to include as its parents, EC provides a *weight function* that assigns a weight to different tipsets. The fork choice rule will then consist of choosing the tipset with the heaviest weight. In practice, EC's weight function [1] is a complex function of (1) the number of blocks in the chain and (2) the total amount of storage committed to the chain. Moreover, the total amount of storage is taken far in the past to ensure that everyone agrees on it. Since in our analysis we assume a static model where the set of participants is fixed during the execution of the protocol, we only take into consideration the number of blocks in the chain. We discuss the impact of this simplification in Section 8. Formally, for a tipset $\mathcal{T}$, we denote its weight $w(\mathcal{T})$ and have:

$$w(\mathcal{T}) = \sum_{\mathcal{T}_i \in \mathcal{C}[\mathcal{T}]} |\mathcal{T}_i|.$$

In the case of a tie between two chains, a deterministic tie-breaker is used. In practice, the tipset that contains the smallest VRF value is chosen. However, in our analysis we consider a powerful adversary that has the power to decide on ties. See Figure 1b for a visual representation of two tipsets with equal weight.

3.4 Mining Algorithm

We describe the mining algorithm that miners in Filecoin run continuously. At each epoch $i > 0$ each participant with key pair $(\mathsf{sk}, \mathsf{pk})$ performs the following:

1. Fetch the drand value for epoch i and verify eligibility by checking

   $$G_{\mathsf{sk}}(\text{drand}_i) \overset{?}{\leq} \mathsf{target},$$

   where target is chosen such that on expectation m leaders are elected (with $m = 5$ in the current implementation).
2. If elected leader, create a block as follows:
   - Choose the tipset with the highest weight (i.e., the most blocks) and reference it as the block's parent.
   - Include a proof of eligibility (i.e., the VRF value: $\mathsf{VRF.Proof}_{\mathsf{sk}}(\mathsf{drand}_i) = (y, p)$), a $\mathsf{WinningPost}$ to prove storage maintenance, as well as the payload.
3. Broadcast the block newly created.

In parallel, whenever they receive a new block in epoch i, participants verify its validity and, if it is valid, add it to their blockDAG. A block is valid if and only if:

1. The election proof (y, p) is valid, i.e., $\mathsf{VRF.Verify}_{\mathsf{pk}}(\mathsf{drand}_i, y, p) = 1$ and $y \leq \mathsf{target}$.
2. The $\mathsf{WinningPost}$ is valid (details omitted).
3. All the transactions in the payload are valid (details omitted).
4. All its parent blocks form a valid tipset, i.e.:
   - They all belong to the same epoch.
   - They all have the same parents.
   - They are all valid blocks.

We analyze the backbone of EC in a static setting and hence omit some details of the protocol. For example, in practice, the leader election mechanism uses a *lookback parameter*, meaning that only participants who pledged their storage sufficiently in the past are eligible for block creation. Because we consider a flat and static model, these details are not relevant to our analysis.

4 Security Definitions

Security properties. We consider the standard security properties of *robust transaction ledgers* defined for blockchain systems [16, 14]. We start by defining a transaction ledger and confirmed transactions in the ledger.

▶ **Definition 1** (Transaction ledger generated by a chain $\mathcal{C}$). *Given a chain $\mathcal{C}$, a transaction ledger $\mathcal{L}$ generated by C is a deterministic, totally-ordered and append-only list of transactions. In particular, if $\mathcal{C}_1$ is a prefix of $\mathcal{C}_2$, then $\mathcal{L}_1$ generated by $\mathcal{C}_1$ is a prefix of $\mathcal{L}_2$ generated by $\mathcal{C}_2$.*

For example, one way to generate a transaction ledger from a chain $\mathcal{C}$ is to order the transactions from $\mathcal{C}$ by order of chronological appearance (i.e., epoch number where they appeared in the chain) and lexicographical order. Any deterministic rule is however valid and we leave this unspecified.

▶ **Definition 2** (Confirmed transaction parameterized by $\tau \in \mathbb{R}$). *If a transaction* tx *in the ledger appears in a block which is mined in epoch $j \leq i - \tau$, then* tx *is said to be τ-*confirmed *in epoch i.*

Our goal is to generate a transaction ledger that satisfies *persistence* and *liveness* as defined in [16, 14]. Together, persistence and liveness guarantee a robust transaction ledger; transactions will be adopted to the ledger and be immutable.

▶ **Definition 3** (Robust transaction ledger from [16, 14]). *A blockchain protocol Π maintains a robust transaction ledger if the generated ledger satisfies the following two properties:*

- *(Persistence) Parameterized by $\tau \in \mathbb{R}$. If a transaction* tx *becomes τ-confirmed at epoch i in the view of one honest node, then* tx *will be at least τ-confirmed in the same position in the ledger by all honest nodes for every epoch $k \geq i$.*
- *(Liveness) Parameterized by $u \in \mathbb{R}$, if a transaction* tx *is received by all honest nodes at epoch i, then after epoch $i + u$ all honest nodes will contain* tx *in the same place in the ledger forever.*

Notations. We then define random variables and stochastic processes of interest and their properties.

Let α and β be the collective fraction of storage power controlled by honest nodes and malicious nodes, respectively ($\alpha + \beta = 1$). We follow the notations of [11]. Let $H[r]$ and $Z[r]$ be the number of blocks mined by the honest nodes and by the malicious nodes in epoch r, then $H[r]$, $Z[r]$ are independent Poisson random variables with means $(1 - \beta)m$ and βm respectively [1] (the value of the target parameter is chosen to ensure this). In addition, the random variables $\{H[0], H[1], \cdots\}$ and $\{Z[0], Z[1], \cdots\}$ are independent of one another, since the value provided by drand to feed the leader election is random. We now define the auxiliary random variables $X[r]$ and $Y[r]$ as follows. If at epoch r an honest node mines at least one block (i.e., $H[r] \geq 1$), then $X[r] = 1$ and epoch r is called a *successful* epoch, otherwise $X[r] = 0$. If at epoch r honest nodes mine exactly one block (i.e., $H[r] = 1$), then $Y[r] = 1$ and epoch r is called a *uniquely successful* epoch, otherwise $Y[r] = 0$. Epoch r is called an *isolated successful* epoch if it further satisfies that there is no honest block in epoch $r - 1$ (i.e., $H[r - 1] = 0$ and $Y[r] = 1$). Further, $X[r_1, r_2]$ and $Y[r_1, r_2]$ are the number of successful and uniquely successful epochs, respectively, in the interval $(r_1, r_2]$, and $H[r_1, r_2]$ and $Z[r_1, r_2]$ are the number of blocks mined by honest nodes and by the adversary respectively in the interval $(r_1, r_2]$.

In EC, chains may have equal weights. For simplicity and generality, we assume tie-breaking always favors the adversary. This means that the persistence will be broken as long as there are two sufficiently long forks with equal weights.

Given a chain of tipsets $\mathcal{C}$, let $\mathcal{C}[r]$ be the chain truncated up to blocks in epoch r. Further, let $w(\mathcal{C})$ be the weight of $\mathcal{C}$. Let $W_{\max}[r]$ and $W_{\min}[r]$ be the maximum and minimum weights of chains adopted by honest nodes at the end of epoch r. Then, by our network model, we have:

$$W_{\min}[r] \leq W_{\max}[r] \leq W_{\min}[r + 1]. \tag{1}$$

Even if some honest nodes' chains are "behind" in epoch r, by our re-broadcast mechanism, their view for epoch r will catch up with the rest of the honest nodes in epoch $r + 1$. Furthermore, honest participants always extend the heaviest chain they are aware of, hence the inequality above.

We also have the following minimum honest chain growth property, which is essential to our proof. For $t \geq r+1$,

$$W_{\min}[t] \geq W_{\min}[r+1] + X[r+1,t] \geq W_{\max}[r] + X[r+1,t]. \tag{2}$$

This inequality again follows from the fact that honest participants always extend the heaviest chain they know of. However, it could be that different honest participants have different views and thus create blocks on different chains, hence why we consider X in the inequality above and not H.

5 Security Proof

In this section, we prove our main theorem, Theorem 4 stated below, parameterized by the security parameter κ. The proof will proceed in multiple steps. We extend the technique of Nakamoto blocks developed in [14]. We first define the notion of Nakamoto epochs in EC and prove that the honest blocks mined in Nakamoto epochs remain in the heaviest chain forever. Then we show that Nakamoto epochs exist and appear frequently regardless of the adversarial strategy. Straightforwardly, the protocol satisfies liveness and persistence: transactions can enter the ledger frequently through the Nakamoto epochs, and once they enter, they remain at a fixed location in the ledger.

▶ **Theorem 4.** *If $\beta m < 1 - e^{-(1-\beta)m}$, then EC generates a robust transaction ledger that satisfies* persistence *(parameterized by $\tau = \kappa$) and* liveness *(parameterized by $u = \kappa$) in Definition 3 with probability at least $1 - e^{-\Omega(\kappa^{1-\epsilon})}$, for any $0 < \epsilon < 1$.*

5.1 Nakamoto epochs

Let us define the events:

$$E_{rs} = \{\text{event that } Z[r-1,t] < X[r+1,t] \text{ for all } t \geq s\},$$

$$F_s = \bigcap_{0 \leq r \leq s-2} E_{rs},$$

$$U_s = \{\text{event that epoch } s \text{ is an isolated successful epoch}\} = \{H[s-1] = 0, Y[s] = 1\},$$

and

$$G_s = F_s \cap U_s.$$

We will call epoch s a *Nakamoto epoch* if the event G_s occurs. And we have the following lemma.

▶ **Lemma 5.** *If epoch s is a Nakamoto epoch, then the unique honest block mined in epoch s is contained in any future chain $\mathcal{C}[t]$, $t \geq s$.*

Proof. Let b_s be the unique honest block mined in epoch s. We will argue by contradiction. Suppose G_s occurs and let $t \geq s$ be the smallest t such that b_s is not contained in $\mathcal{C}[t]$, an honest chain adopted by some honest node at the end of epoch t. Let b_r, mined in epoch r, be the last honest block on $\mathcal{C}[t]$ (which must exist, because the genesis block is by definition honest). If $r > s$, then $\mathcal{C}[r-1]$ is the prefix of $\mathcal{C}[t]$ before block b_r, and does not contain b_s (because $\mathcal{C}[r-1]$ is a prefix of $\mathcal{C}[t]$) contradicting the minimality of t. So b_r must be mined before or in epoch s. Since epoch s is an isolated successful epoch, we further know

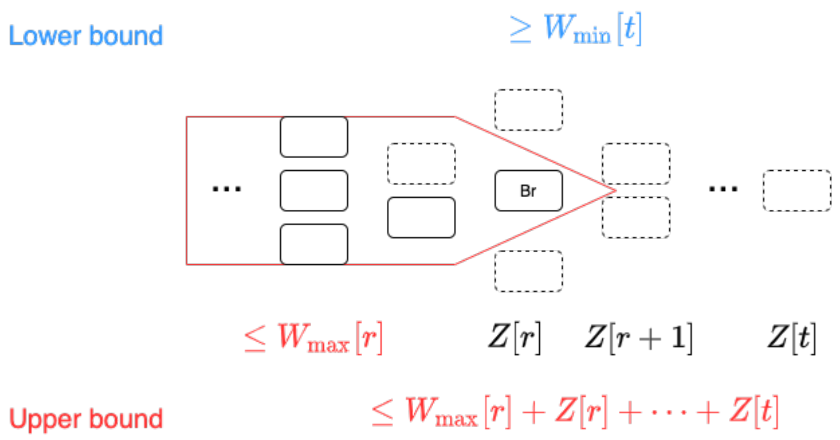


Figure 2 Upper bound and lower bound of the weight of $\mathcal{C}[t]$ in the proof of Lemma 5. Blocks with dotted lines are adversarial blocks. The parent links are omitted for readability; each block has all blocks from the previous epoch as parents.

that $r \leq s - 2$. The part of $\mathcal{C}[t]$ after block b_r must consist of all malicious blocks by the definition of b_r. Note that this may also include malicious blocks in epoch r (i.e., *headstart* of the adversary). Hence, we have an upper bound for the weight of $\mathcal{C}[t]$.

$$w(\mathcal{C}[t]) \leq W_{\max}[r] + Z[r-1,t] < W_{\max}[r] + X[r+1,t], \tag{3}$$

where the first inequality is illustrated in Figure 2, and the second inequality follows from the fact that event F_s occurs. We also have a trivial lower bound: $w(\mathcal{C}[t]) \geq W_{\min}[t]$. Therefore, we have

$$W_{\min}[t] < W_{\max}[r] + X[r+1,t], \tag{4}$$

which contradicts the minimum honest chain growth property (Eqn. 2). ◀

Note that Lemma 5 implies that if G_s occurs, then the entire chain leading to the unique honest block mined in epoch s from the genesis is stabilized after epoch s.

5.2 Occurrence of Nakamoto epochs

Although the existence of Nakamoto epochs ensures that the block at this epoch will be finalized, i.e., it will appear in every honest future chain, the question now remains whether Nakamoto epochs exist at all and, if so, at what frequency they appear. We start answering this question by proving in the next lemma that Nakamoto epochs have a strictly positive probability of happening, i.e., the probability of each epoch being a Nakamoto epoch is strictly positive. Due to space limitations, the detailed proof can be found in Section 5.2 of the full version [29].

▶ **Lemma 6.** *If $\beta m < 1 - e^{-(1-\beta)m}$, then there exists $p > 0$ such that $P(G_s) \geq p$ for all s.*

5.3 Waiting time for Nakamoto epochs

We have established the fact that the event G_s has $P(G_s) \geq p > 0$ for all s. But how long do we need to wait for such an epoch to occur? We answer this question in the following lemma, wherein we provide a bound on the probability that in a interval $(j, j+k]$ of k consecutive epochs, there are no Nakamoto epochs, i.e., a bound on:

$$q(j, j+k] := P\Big(\bigcap_{s=j+1}^{j+k} G_s^c\Big),$$

where G_s^c is the complement of G_s.

▶ **Lemma 7.** *If $\beta m < 1 - e^{-(1-\beta)m}$, then there exist constants α, A so that for all $j, k \geq 0$,*

$$q(j, j+k] \leq A \exp(-\alpha\sqrt{k}). \tag{5}$$

Proof. Following the definition in Lemma 6, let

$$\hat{B}_{rt} = \text{event that } Z[r-1, t] \geq X[r+1, t].$$

Similar to the calculation in Lemma 6, we have

$$P(\hat{B}_{rt}) \leq A_1 e^{-\alpha_1 \varepsilon^2 (t-r)} \tag{6}$$

for some positive constants A_1, α_1 independent of r, t.

Also we have

$$G_s^c = F_s^c \cup U_s^c = \bigcup_{(r,t):r \leq s-2, t \geq s} \hat{B}_{rt} \cup U_s^c. \tag{7}$$

Divide $(j, j+k]$ into $\sqrt{k}$ sub-intervals of length $\sqrt{k}$ (assuming $\sqrt{k}$ is a integer), so that the i-th sub-interval is:

$$\mathcal{J}_i := [j + 1 + (i-1)\sqrt{k}, j + i\sqrt{k}].$$

Now look at the first, fourth, seventh, etc sub-intervals, i.e. all the $i = 1 \mod 3$ sub-intervals. Introduce the event that in the ℓ-th ($1 \mod 3$) sub-interval ($\mathcal{J}_{3\ell+1}$), a pure adversarial chain that is rooted at a honest block (or more accurately a tipset including at least one honest block) mined in that sub-interval ($\mathcal{J}_{3\ell+1}$) or in the previous ($0 \mod 3$) sub-interval ($\mathcal{J}_{3\ell}$) catches up with a honest block in that sub-interval ($\mathcal{J}_{3\ell+1}$) or in the next ($2 \mod 3$) sub-interval ($\mathcal{J}_{3\ell+2}$).

Formally,

$$C_\ell = \bigcap_{s \in \mathcal{J}_{3\ell+1}} \bigcup_{(r,t):r \in \mathcal{J}_{3\ell} \cup \mathcal{J}_{3\ell+1}, r \leq s-2, t \geq s, t \in \mathcal{J}_{3\ell+1} \cup \mathcal{J}_{3\ell+2}} \hat{B}_{rt} \cup U_s^c.$$

Note that for distinct ℓ, the events C_ℓ's are independent since $\hat{B}_{rt}$'s in different C_ℓ's do not have overlap (the $\mathcal{J}$ intervals were cut specifically for this purpose). Also, we have

$$P(C_\ell) \leq P(\text{no Nakamoto epoch in } \mathcal{J}_{3\ell+1}) = 1 - p < 1 \tag{8}$$

by Lemma 6.

Introduce the atypical events:

$$\begin{aligned} B &= \bigcup_{(r,t):r\in(j,j+k] \text{ or } t\in(j,j+k],r<t,t-r\geq\sqrt{k}} \hat{B}_{rt}\ , \text{ and} \\ \tilde{B} &= \bigcup_{(r,t):r\leq j,j+k<t} \hat{B}_{rt}\ . \end{aligned}$$

The events B and $\tilde{B}$ are the events that an adversarial chain catches up with an honest block far ahead (more than $\sqrt{k}$ epochs).

By (6) and an union bound we have that

$$\begin{aligned} & P(B) \\ \leq\ & \sum_{(r,t):r\in[j+1,j+k] \text{ or } t\in[j+1,j+k],r<t,t-r\geq\sqrt{k}} A_1 e^{-\alpha_1\varepsilon^2(t-r)} \\ \leq\ & \sum_{r=j+1}^{j+k}\Big(\sum_{t=r+\sqrt{k}}^{\infty} A_1 e^{-\alpha_1\varepsilon^2(t-r)}\Big) + \sum_{t=j+1}^{j+k}\Big(\sum_{r=0}^{t-\sqrt{k}} A_1 e^{-\alpha_1\varepsilon^2(t-r)}\Big) \\ \leq\ & 2k\frac{A_1 e^{-\alpha_1\varepsilon^2\sqrt{k}}}{1-e^{-\alpha_1\varepsilon^2}}, \end{aligned}$$

and

$$\begin{aligned} P(\tilde{B}) &\leq \sum_{(r,t):r\leq j,t>j+k} A_1 e^{-\alpha_1\varepsilon^2(t-r)} \\ &\leq \sum_{r=0}^{j}\Big(\sum_{t=j+k+1}^{\infty} A_1 e^{-\alpha_1\varepsilon^2(t-r)}\Big) \\ &= \sum_{r=0}^{j}\frac{A_1 e^{-\alpha_1\varepsilon^2(j+k+1-r)}}{1-e^{-\alpha_1\varepsilon^2}} \\ &\leq \frac{A_1 e^{-\alpha_1\varepsilon^2(k+1)}}{(1-e^{-\alpha_1\varepsilon^2})^2}. \end{aligned}$$

Now, we have:

$$\begin{aligned} & q(j,j+k] \\ \leq\ & P(\text{no Nakamoto epoch in } \textstyle\bigcup_{\ell=0}^{\sqrt{k}/3}\mathcal{J}_{3\ell+1}) \\ \leq\ & P(\text{no isolated successful epoch in } \textstyle\bigcup_{\ell=0}^{\sqrt{k}/3}\mathcal{J}_{3\ell+1}) + P(B) + P(\tilde{B}) + P(\displaystyle\bigcap_{\ell=0}^{\sqrt{k}/3} C_\ell) \\ =\ & e^{-\Omega(k)} + P(B) + P(\tilde{B}) + (P(C_\ell))^{\sqrt{k}/3} \qquad (9) \\ \leq\ & e^{-\Omega(k)} + 2k\frac{A_1 e^{-\alpha_1\varepsilon^2\sqrt{k}}}{1-e^{-\alpha_1\varepsilon^2}} + \frac{A_1 e^{-\alpha_1\varepsilon^2(k+1)}}{(1-e^{-\alpha_1\varepsilon^2})^2} + (P(C_\ell))^{\sqrt{k}/3} \\ \leq\ & A\exp(-\alpha\sqrt{k}) \qquad (10) \end{aligned}$$

for some positive constants A and α. The equality (9) is due to the independence of C_ℓ's and the inequality (10) is due to (8). Hence the lemma follows. ◀

We can also tighten the exponent, but at the cost of larger constants in the bound. The proof of the following lemma is almost verbatim identical with the proof of Lemma 7, and its detailed explanation can be found in Appendix C of the full version [29].

▶ **Lemma 8.** *If $\beta m < 1 - e^{-(1-\beta)m}$, then there exist constants $\alpha_\epsilon, A_\epsilon$ so that for all $j, k \geq 0$,*

$$q(j, j+k] \leq A_\epsilon \exp(-\alpha_\epsilon k^{1-\epsilon}), \tag{11}$$

for any $0 < \epsilon < 1$.

5.4 Persistence and liveness

Equipped with all the previous lemmas, we can now prove the persistence and liveness properties of EC for $\beta m < 1 - e^{-(1-\beta)m}$.

Proof of Theorem 4. Suppose current epoch is r. Then by Lemma 8, with probability at least $1 - e^{-\Omega(\kappa^{1-\epsilon})}$, there is at least one Nakamoto epoch in the interval $(r - \kappa, r]$. Let epoch $s \in (r - \kappa, r]$ be a Nakamoto epoch. Then by Lemma 5, the chain up to epoch $s - 1$ is permanent since the unique honest block in epoch s never leaves the heaviest chain. Hence EC is persistent with probability at least $1 - e^{-\Omega(\kappa^{1-\epsilon})}$. The liveness of EC is simply a consequence of the frequent occurrence of Nakamoto epochs. Particularly, for each honest transaction, either it will be included by an honest block B in a Nakamoto epoch or it has already been included by B's ancestors. ◀

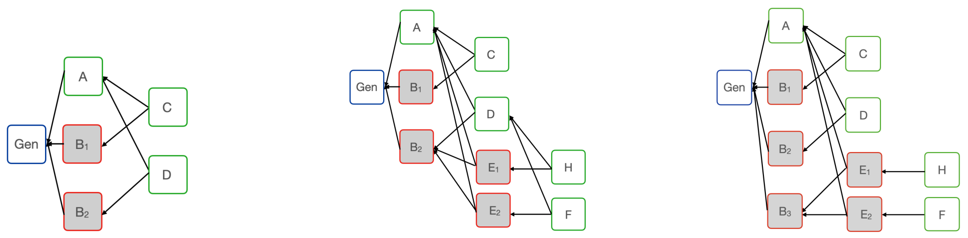
6 n-split Attack

In order to confirm whether an adversary with power $\beta m \geq 1 - e^{-(1-\beta)m}$ can indeed break the persistence and liveness properties of the system, we consider the following n-split attack.

6.1 Attack description

The attacker tries to split the honest participants among n chains such that in each epoch, at most one honest block is added to each chain (i.e., no two honest players mine on the same chain). To do this, the attacker creates n copies of one of its block (each copy has the same election proof, but different payloads) and sends one different block to each of the n honest players; see illustration in Figure 3a. To maintain the split for a long period, the adversary must repeat the attack at every epoch where at least one honest block is mined. In this case, the weight of the chain of each honest player will increase by two: one honest block and one adversarial block. For example in Figure 3a, since blocks C and D are both honest (i.e., created by honest miners), by the next epoch, epoch 3, all the participants will have received them and use the deterministic tie breaker to all decide to mine on the same tipset, say {D}. Hence the adversary must create equivocating blocks in epoch 2 as well in order to ensure that in epoch 3, honest miners all choose different tipsets to append their block to. In Figure 3b, the adversary sends equivocating blocks E_1 and E_2 to prevent blocks H and F from being appended to the same chain. These figures include only two honest blocks at epoch 2 and 3 for clarity. In practice, the adversary will create as many equivocating blocks as there are honest miners to ensure that everyone sees a different block and that no two honest participants mine on the same tipset.

Whenever there is no honest block mined in an epoch, the attacker does nothing. In this case, the weight of the chain of each honest player will not increase. Meanwhile, the attacker also reuses all its blocks to build a private chain, i.e., a chain that it does not broadcast to other participants and that does not include any honest blocks. The expected chain growth of the adversary's private chain is βm. The weight of the honest chain increases by two if there is at least one honest block mined in an epoch (which happens with probability



(a) The adversary sends two different blocks B_1, B_2 in epoch 1 such that in epoch 2, honest blocks C, D have different parents and hence cannot be included in a tipset. The honest power is thus split between different tipsets' chains.

(b) The adversary keeps the network split in epoch 3 by creating two equivocating blocks: E_1 and E_2 in epoch 2. In epoch 3, honest blocks H and F are mined on two different tipsets.

(c) If ties are always broken in favor of the adversary, the adversary can instead create another block B_3 in epoch 1. In epoch 2, blocks E_1 and E_2 are preferred to C and D, hence in epoch 2, all the tipsets' chains increase by only one block. The attack is repeated in the next epoch, as long as the adversary has enough blocks to create a fork as heavy as the honest chains.

Figure 3 n-split attack. Each block filled in grey is an equivocating block, meaning they were created by the adversary using the same leader election proof in one epoch. Each green block is an honest block.

$1 - e^{-(1-\beta)m}$), and 0 otherwise (which happens with probability $e^{-(1-\beta)m}$). Hence, the expected chain growth of the honest chain is $2(1-e^{-(1-\beta)m})$. Therefore, this attack succeeds with non-negligible probability when $\beta m > 2(1 - e^{-(1-\beta)m})$, i.e., when the adversarial chain grows at a higher rate than the honest split chains. Rather than specifying the exact success probability, we demonstrate that it remains constant, independent of the confirmation depth τ, as defined in Definition 2. Let L be adversarial lead, i.e., the adversary has a lead of L additional private blocks over the public heaviest chain. Suppose that with probability p_{L_0}, the initial lead before the attack starts is L_0. Although p_{L_0} decreases with L_0, it remains non-zero because there's a chance the adversary could mine L_0 blocks before the honest nodes mine any block. Note that in the n-split attack, as long as the adversarial lead $L > 0$, the adversary can invariably split the honest nodes across n chains. Therefore, the adversarial private chain will grow faster than the honest public chain when $\beta m > 2(1 - e^{-(1-\beta)m})$. According to the standard random walk (with drift) theory [15], L goes to 0 only with a probability of $e^{-O(L_0)}$. This implies that the n-split attack succeed with probability at least $p_{L_0}(1 - e^{-O(L_0)})$, for any value of the confirmation depth τ.

Some numerical results: for $m = 3$, we have $\beta > 0.512$; for $m = 5$, we have $\beta > 0.382$; and $\beta > 0.284$ for $m = 7$. Approximately, $\beta \gtrsim 2/m$. Intuitively, with this attack, the threshold is inversely proportional to m, as with a bigger m, the adversary is elected leader more often and thus has more opportunities to keep the network split. If the adversary were elected leader on a less regular basis, it would be harder to keep sending equivocating blocks and thus keep the network split for longer. Note that this threshold applies specifically to the attack described above, and it differs from the security threshold identified in Theorem 4. This is because different thresholds may exist for different attacks.

Following the standard model of longest-chain analysis [16, 13, 14], we give the power of tie-breaking to the attacker (i.e., tie-breaking always favors the attacker's block). Recall that the same assumption is made in our model (Section 2) and proof (Section 5). For example in Figure 3c, if we assume that the adversarial blocks (in red) will always be favored in the case of two chains with the same number of blocks, then the adversary does not need to create a block on the same tipset as honest blocks (as in Figure 3b). The adversary will instead create another block B_3 and mine yet on another tipset than the honest participant in epoch 2. In epoch 3 the adversarial chains ending in tipsets $\{E_1\}$ and $\{E_2\}$ are preferred over $\{C\}$ or $\{D\}$, hence honest blocks H and F are mined on different forks and each fork's weight increased by only one in epoch 2, as opposed to 2 in Figure 3b. By repeating this attack at each epoch, the weight of the chain of each honest player will only increase by one when there is at least one honest block mined. Following the same argument as above, this attack succeeds with non-negligible probability when $\beta m > (1 - e^{-(1-\beta)m})$, i.e., when the adversarial chain grows at a higher rate than the honest split chains. We notice that this now matches the security threshold in Theorem 4, hence proving that $\beta m = 1 - e^{-(1-\beta)m}$ is the tight threshold of the protocol in our security model as defined in Section 2. Indeed, Theorem 4 proves that no adversary below this threshold can break the security of the EC, and the n-chain split attack just described proves that an adversary above this threshold can indeed break the persistence and liveness of the EC.

Some numerical results: for $m = 3$, we have $\beta \simeq 0.293$; for $m = 5$, we have $\beta \simeq 0.196$; and $\beta \simeq 0.143$ for $m = 7$. Approximately, $\beta \simeq 1/m$. As remarked before, the threshold is inversely proportional to the number of leaders elected as, intuitively, being elected more often gives more opportunities to an adversary.

6.2 Discussion

Rationality of the attack. We note that this attack is detectable as everyone can see that blocks with the same proof of eligibility but different payload were created. In practice, this behaviour is slashable in Filecoin [1]. However, in Filecoin an adversary has the ability to spread its storage over multiple identities, i.e., create multiple identities that each possesses one unit of storage. For example in Filecoin the minimum unit of storage that can be pledged to the chain is 32 GiB. As of August 2023 the total storage pledged to the chain is around 11 EiB [6], hence an adversary that possesses 20% of the total power, i.e., 2.2 EiB could potentially "spread" its storage over $\frac{2.2\times10^{18}}{32\times10^{9}} \simeq 10^8$ different identities, that each possesses 32 GiB of storage. At each epoch, except with extremely small probability, the adversary will have a new "identity" elected to create a block (it is very unlikely for a miner with 32GiB of storage out of 11 EiB to be elected twice in a row). Assuming that each identity gets slashed and removed from the list of participants after equivocation, after performing the attack over 1000 epochs, the adversary will still have $9.9999 \cdot 10^7$ identities left out of 10^8 and will only be slashed $\frac{10^3}{10^8} = 10^{-5}$ of its total collateral. In this analysis we considered orders of magnitude rather than exact numbers, so for simplicity we counted only the "real power" and did not account for the "boosted adjusted power" that can be gained through the FIL+ program [5]. Note that it is not possible for any honest miner to know which identities belong to the adversary before the equivocation. Hence excluding equivocating participants from the protocol is not sufficient to prevent the attack.

Furthermore, we note that for the adversary to be slashed, a special transaction, *a fraud proof* transaction must be submitted on-chain by any participant. An adversary that is able to continually exclude honest blocks as is the case with the n-split attack may thus in practice never be slashed as no honest participant will get the opportunity to include the slashing transaction on-chain. This is why even when considering incentives and the slashing mechanism in place, this attack is still rational.

Network control. In this attack, we assumed a powerful adversary that not only has the power to break ties in its favor, but has also full control of the network, as specified in Section 2.2. However we remark that for this attack to work, an adversary only needs limited power over the network. Specifically, the adversary needs to be directly connected to every participants but does not need to control the propagation time between honest participants, as we illustrate now.

In Filecoin honest miners will stop accepting blocks for an epoch after a cutoff time. For the split to happen the adversary could send each block $B_1, \ldots, B_n$ to each different participants $1, \ldots, n$ right before the cutoff, i.e., participant i receives block B_i from the adversary just before the cutoff time, ensuring block B_i is accepted by participant i. The adversary would need to know the propagation time between itself and each participant to do so, however this is easy to estimate. Since participant i, receives B_i just before the cutoff time, whatever the propagation delay between i and another honest participant j is, the adversary is guaranteed that j will received B_i from i *after* the cutoff time and hence that any honest miner $j \neq i$ will not accept B_i.

Furthermore we note that when participant j receives block B_i, after the cutoff, j will detect the equivocation as j already received block B_j from the adversary. However at that point, j has already created its block that includes B_j as a parent, hence it is too late for j to discard B_j due to equivocation. The mitigation that we propose in Section 7.2 changes this.

7 Mitigations

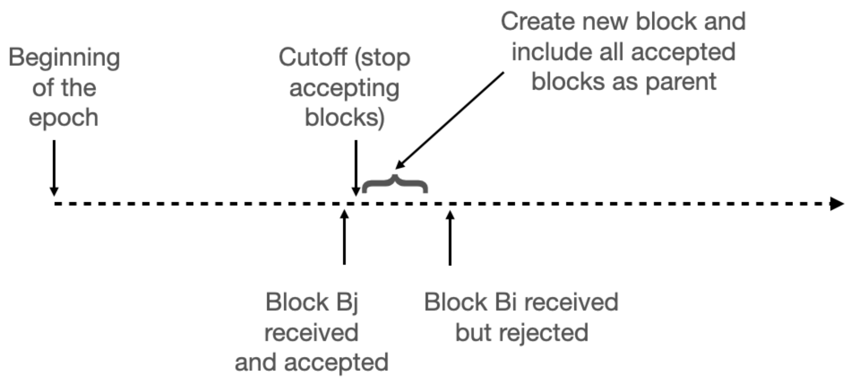
We propose two possible mitigations to the n-split attack described in Section 6 which could also help increase the security threshold of EC.

7.1 Replace EC by the Longest-chain Protocol in SPC

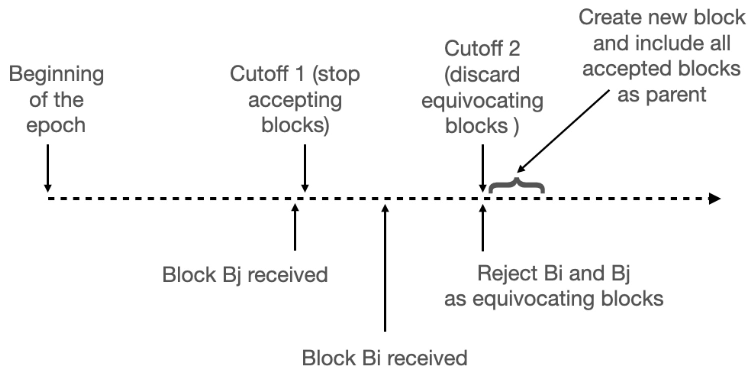
One solution to the n-split attack is to remove the notion of tipsets and instead change EC to the longest-chain protocol (i.e., Ouroboros family of protocols [19, 13, 9] in the proof-of-stake setting), where one block has exactly one parent. In the longest-chain protocol, the effort of an adversary to split the network would have much less impact on the overall security of the protocol since each chain can increase by one block at each epoch at most anyway. Furthermore, moving to the longest-chain protocol allows for inheritance of all the security properties (e.g., a security threshold of 50%) of all proof-of-stake protocols based on that setting [13, 10, 14]. Dembo et al. [14] indeed showed that in the longest-chain case, the worst attack is the private attack. Hence, the n-split attack described in Section 6, or its variant, would not be the worst attack anymore. However, transitioning from EC to a longest-chain protocol is not a straightforward task. First, it's crucial to understand that merely setting $m = 1$ does not transform EC into a longest-chain protocol, as more than one block can still be added per epoch. Furthermore, our analysis indicates that an EC with $m = 1$ has a security threshold of approximately 43.2%, as opposed to 50% in the longest-chain protocol. Consequently, to enhance the security, the concept of a tipset will need to be eliminated. Implementing such a change, however, would necessitate a hard fork.

7.2 Consistent Broadcast

Another solution is to use a form of consistent or reliable broadcast [12, 18]. This type of broadcast prevents an adversary from equivocating (i.e., creating two blocks with the same leader election proof but different contents). The consistent broadcast consists in


Beginning of the epoch
Cutoff (stop accepting blocks)
Create new block and include all accepted blocks as parent
Block Bj received and accepted
Block Bi received but rejected

(a) Current case.


Beginning of the epoch
Cutoff 1 (stop accepting blocks)
Cutoff 2 (discard equivocating blocks)
Create new block and include all accepted blocks as parent
Block Bj received
Block Bi received
Reject Bi and Bj as equivocating blocks

(b) Case with a (simplified) consistent broadcast.

Figure 4 The dashed arrow represents the arrow of time. The different cutoff and arrival time are marked with vertical arrows. The adversary ensures that j receives B_j right before the cutoff so j accepts B_j. In the first case, by the time j sees an equivocation, it is too late as B_j was already included as a parent. In the second case, assuming j receives B_i before the second cutoff, then j will discard B_j and not include it as a parent.

adding a second cutoff to the cutoff discussed in Section 6.2. Specifically, it ensures that after participant j received B_j from the adversary (before the first cutoff), j will wait for a "second" cutoff before forming its block and including B_j as a parent. When j receives equivocating block B_i from the adversary, j will detect the equivocation and decide not to include B_j (neither B_i), hence the attack is mitigated. This is illustrated in Figure 4. The epoch in Filecoin is thus split as follows: during the first period, participants will store every valid block received in their "pending blocks" set. In the second period, after the first cutoff and before the second cutoff, every new valid block received will be stored in the set of "rejected blocks" for that epoch. In the third period, after the second cutoff, participants will compare the set of pending blocks and rejected blocks, if they detect equivocating blocks, they are removed from the pending set. Every block that is left in the pending set will then be included in the tipset. This implementation as it is simple and backward compatible (i.e., only requires a soft-fork) although it assumes synchronicity. With consistent broadcast, however, the adversary is still able to split the network in two ways. First it could do so by ensuring that only a fraction of the honest nodes accept its block (i.e., the network will be split between the nodes that accept the adversarial block vs those that do not). The second way in which the network could be split is if the adversary is elected more than once, say ℓ times in one epoch (here we assume that the adversary controls many different participants). Then the adversary could similarly ensure that for each block it is able to create (with different election proofs), only some of the nodes accept it. The network is then split in 2^ℓ ways (all the combination of accept vs reject for each of the ℓ blocks). We hypothesize that, under a non-equivocating adversary, the security threshold of EC with $m = 5$ is approximately 40%. Intuitively, with this level of power, the adversary creates fewer than two blocks per epoch on average; furthermore, without the ability for equivocation, it is impossible for the adversary to maintain two chains of equal weights over numerous epochs. We leave a formal proof as future work.

8 Limitations and Future Work

For practical reasons, this work made a few simplifying assumptions. We discuss them here.

Incentive consideration. This work considers the classic model of honest vs malicious participants and does not address the rationality of participants. A formal study of incentive compatibility is also important for understanding the security of EC. However, we leave this for future work. Assuming a fully malicious adversary that is willing to lose money to attack the system, a scenario we consider in this paper, makes for a stronger proof than assuming a rational adversary. In Section 6.2, we discussed why it is realistic to consider an irrational adversary for the n-split attack we proposed, as slashing may not always be possible if the adversary has the ability to censor transactions. It still remains to show that the honest strategy is compatible with a rational strategy even in the presence of an adversary. We leave this for future work.

Weight function. In our analysis, we only took into consideration the number of blocks in the chain for the weight function. We leave as future work an analysis that also considers the total storage, as specified in Expected Consensus [1]. Specifically we believe that a complex weight function allows for more vectors of attack and that an adversary could use this to try to blow the weight of its own chain. For example, the adversary could remove its storage from the main chain and thus decrease the weight of the main chain, while privately creating an alternative chain that would be heavier because it has more storage pledged. The mechanisms for maintaining and removing storage are, however, complex and ignored in this work. For simplicity, we thus consider the weight of a tipset to simply be equal to the number of blocks referenced in its blockDAG.

9 Conclusion

In this paper we presented a formal analysis of Expected Consensus, a sub-protocol of Filecoin's Storage Power Consensus, and we proposed two concrete ways to improve SPC's security. One of our mitigations, using consistent broadcast, is currently being implemented as a Filecoin Improvement Proposal. It remains an open problem to quantify the new security threshold of EC with this fix, although our proofs remain valid in this case, hence the security threshold is at least such that $\beta m < 1 - e^{-(1-\beta)m}$ as proved in Section 5. Furthermore, we made many simplifying assumptions in this work. It would be interesting to relax these in the future work; e.g., by extending this proof to the dynamic and asynchronous case, considering the more complex variant of the weight function or incorporating incentives.

References

1 Filecoin Spec. https://spec.filecoin.io/.

2 A Guide to Filecoin Storage Mining. https://filecoin.io/blog/posts/a-guide-to-filecoin-storage-mining/. Accessed: 2023-08-02.

3 Coin Market Cap. https://coinmarketcap.com/.

4 Drand. http://https://drand.love/. Accessed: 2022-08-30.

5 Filecoin Plus. https://docs.filecoin.io/basics/how-storage-works/filecoin-plus/.

6 Filfox - Filecoin Explorer. https://filfox.info/en.

7 FIP-0051: Improving EC security with Consistent Broadcast. https://github.com/filecoin-project/FIPs/blob/master/FRCs/frc-0051.md.

8 Sarah Azouvi and Marko Vukolić. Pikachu: Securing PoS blockchains from long-range attacks by checkpointing into Bitcoin PoW using Taproot. *arXiv preprint arXiv:2208.05408*, 2022.

9 Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930, 2018.

10 Vivek Bagaria, Amir Dembo, Sreeram Kannan, Sewoong Oh, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Proof-of-stake longest chain protocols: Security vs predictability. *arXiv preprint arXiv:1910.02218*, 2019.

11 Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602. ACM, 2019.

12 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.

13 Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

14 Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and Nakamoto always wins. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 859–878, 2020.

15 William Feller. An introduction to probability theory and its applications. Technical report, Wiley series in probability and mathematical statistics, 3rd edn.(Wiley, New …, 1971.

16 Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

17 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

18 Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Dragos-Adrian Seredinschi, and Yann Vonlanthen. Scalable Byzantine reliable broadcast (extended version). *arXiv preprint arXiv:1908.01738*, 2019.

19 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

20 Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. A better method to analyze blockchain consistency. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 729–744, 2018.

21 Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

22 Satoshi Nakamoto and A Bitcoin. A peer-to-peer electronic cash system. *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, 4(2), 2008.

23 Christopher Natoli and Vincent Gramoli. The balance attack against proof-of-work blockchains: The r3 testbed as an example. *arXiv preprint arXiv:1612.09426*, 2016.

24 Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology–EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30–May 4, 2017, Proceedings, Part II*, pages 643–673. Springer, 2017.

25 Rafael Pass and Elaine Shi. The sleepy model of consensus. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 380–409. Springer, 2017.

26 Michael O Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, 1983.

27 Ling Ren. Analysis of Nakamoto consensus. *Cryptology ePrint Archive*, 2019.

28 Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers 19*, pages 507–527. Springer, 2015.

29 Xuechao Wang, Sarah Azouvi, and Marko Vukolić. Security analysis of filecoin's expected consensus in the byzantine vs honest model. *arXiv preprint arXiv:2308.06955*, 2023.

Appendix

A Pseudocode for EC

The main algorithm is presented in Algorithm 1 and the algorithm for block validation is presented in Algorithm 2.

B Concentration Inequalities

▶ **Lemma 9** (Chernoff). *Let $X = \sum_{i=1}^{n} X_i$, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$, and all X_i's are independent. Let $\mu = \mathbb{E}[X] = \sum_{i=1}^{n} p_i$. Then for $0 < \delta < 1$, $\mathbb{P}(X > (1+\delta)\mu) < e^{-\Omega(\delta^2\mu)}$ and $\mathbb{P}(X < (1-\delta)\mu) < e^{-\Omega(\delta^2\mu)}$.*

▶ **Lemma 10** (Poisson). *Let X be a Poisson random variable with rate μ. Then for $0 < \delta < 1$, $\mathbb{P}(X > (1+\delta)\mu) < e^{-\Omega(\delta^2\mu)}$ and $\mathbb{P}(X < (1-\delta)\mu) < e^{-\Omega(\delta^2\mu)}$.*

Algorithm 1 Main algorithm.

```
1: import
2:     drand
3:     ForkChoiceRule
4:     Broadcast
5:     VRF
6:     isValid (Algorithm 2)
7: Parameters:
8:     epochLength
9:     $m$
10:    target                         ▷ Chosen such that $m$ leaders are elected on expectation
11: Init:
12:    epochNumber ← 0
13:    blockDAG←{Genesis Block}
14: upon event time.Now() % epochLength == 0 do          ▷ Beginning of the epoch
15:    epochNumber ← epochNumber +1
16:    seed ← drand(epochNumber)
17:    $(y, p)$ ← $\mathsf{VRF.Proof}_{\mathsf{sk}}$(seed)
18:    if $y \leq$ target then
19:        $\mathcal{T}$ ←ForkChoiceRule(blockDAG)        ▷ Choose the DAG with the most blocks
20:        $\mathcal{B}$ ← CreateBlock($\mathcal{T}$, $(y, p)$, epochNumber,WinningPost payload)
21:        Broadcast($\mathcal{B}$)
22: upon event Receiving block $\mathcal{B}$ do
23:    if isValid($\mathcal{B}$) == 1 then
24:        blockDAG.append($\mathcal{B}$)
```

Algorithm 2 isValid($\mathcal{B}$).

```
1: Input: block $\mathcal{B}$
2: import
3:     drand
4:     isPayloadValid
5:     isStorageValid
6: Parse ($\mathcal{T}$, $(y, p)$, epochNumber,WinningPost payload) ← $\mathcal{B}$
7: seed ← drand(epochNumber)
8: if $\mathsf{VRF.Verify}_{\mathsf{pk}}(\mathsf{seed}, y, p)$ == 0 or $y >$ target then   ▷ Check the election proof
9:     return 0
10: if isPayloadValid(payload)==0 then                   ▷ Check the payload
11:    return 0
12: if isStorageValid(WinningPost) == 0 then            ▷ Check the storage proof
13:    return 0
14: for $\mathcal{B}_i \in \mathcal{T}$ do                ▷ Check validity of parent blocks
15:    if isValid($\mathcal{B}_i$) == 0 then
16:        return 0
17: return 1
```

Tailstorm: A Secure and Fair Blockchain for Cash Transactions

Patrik Keller ✉
Universität Innsbruck, Austria

Ben Glickenhaus
University of Massachusetts Amherst, MA, USA

George Bissias
University of Massachusetts Amherst, MA, USA

Gregory Griffith
Bitcoin Unlimited

Abstract

Proof-of-work (PoW) cryptocurrencies rely on a balance of security and fairness in order to maintain a sustainable ecosystem of miners and users. Users demand fast and consistent transaction confirmation, and in exchange drive the adoption and valuation of the cryptocurrency. Miners provide the confirmations, however, they primarily seek rewards. In unfair systems, miners can amplify their rewards by consolidating mining power. Centralization however, undermines the security guarantees of the system and might discourage users.

In this paper we present Tailstorm, a cryptocurrency that strikes this balance. Tailstorm merges multiple recent protocol improvements addressing security, confirmation latency, and throughput with a novel incentive mechanism improving fairness. We implement a parallel proof-of-work consensus mechanism with k PoWs per block to obtain state-of-the-art consistency guarantees [29]. Inspired by Bobtail [9] and Storm [4], we structure the individual PoWs in a tree which, by including a list of transactions with each PoW, reduces confirmation latency and improves throughput. Our proposed incentive mechanism discounts rewards based on the depth of this tree. Thereby, it effectively punishes information withholding, the core attack strategy used to reap an unfair share of rewards.

We back our claims with a comprehensive analysis. We present a generic system model which allows us to specify Bitcoin, $\mathcal{B}_k$ [29], and Tailstorm from a joint set of assumptions. We provide an analytical bound for the fairness of Tailstorm and Bitcoin in *honest* networks and we confirm the results through simulation. We evaluate the effectiveness of *dishonest* behaviour through reinforcement learning. Our attack search reproduces known optimal strategies against Bitcoin, uncovers new ones against $\mathcal{B}_k$, and confirms that Tailstorm's reward discounting makes it more resilient to incentive layer attacks. Our results are reproducible with the material provided online [30].

Lastly, we have implemented a prototype of the Tailstorm cryptocurrency as a fork of Bitcoin Cash. The client software is ready for testnet deployment and we also publish its source online [23].

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Proof-of-Work, Blockchain, Cryptocurrency, Mining Rewards, Fairness

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.6

Related Version *Extended Version*: `https://arxiv.org/abs/2306.12206`

Supplementary Material *Software (Evaluation)*: `https://github.com/pkel/cpr/tree/aft23` [30]
archived at `swh:1:dir:3ad6764f1d3e29fdf2a8c959581758c46e0a6817`
Software: `https://gitlab.com/georgebissias/BCHUnlimited/-/tree/tailstorm_prototype` [23]
archived at `swh:1:dir:8f9927f6e42866a3f13984232b4004defcdd3a36`

Funding *George Bissias*: Bitcoin Unlimited

Acknowledgements We wish to thank Bitcoin Unlimited for their financial and technical support as well as Michael Fröwis for his review of this work and for the helpful suggestions he provided.

1 Introduction

Proof-of-work (PoW) cryptocurrencies can be thought of as stacked systems comprising four layers. The PoW layer moderates access of weakly identified parties using a mining puzzle: proposing a new block requires finding a small hash. The consensus layer allows all participants to agree on a specific block ordering. The application layer maintains a distributed ledger by writing cryptocurrency transactions into the blocks. Finally, the incentive layer motivates participation in the PoW layer by minting new cryptocurrency for successful miners.

The circular dependencies between the layers result in interdependent failures. Consensus faults are inevitable when individual miners gain too much control [19]. Unreliable consensus enables double spending and erodes confidence in the cryptocurrency. Devaluation of the currency renders the mining rewards worthless as well. Lastly, misaligned incentives encourage centralization among the miners, eventually allowing the strongest one to break consensus.

In this paper, we introduce and analyze Tailstorm, a new cryptocurrency that strengthens two layers of the stack. We employ an innovative incentive mechanism as well as a state-of-the-art consensus mechanism, while retaining the PoW and application layers of Bitcoin. We draw inspiration from the organization of delta blocks in the Storm protocol [4] as well as the use of partial PoW in the Bobtail [9] and $\mathcal{B}_k$ protocols [29].

Bitcoin's consensus [39] uses a *sequential* PoW mechanism where each block references a single parent block. The blocks form a tree and the participants mine new blocks that extend the longest branch according to the *longest chain rule*. Blocks off the longest branch are discarded. Evidently, attackers who possess more than 50 % of the hash rate pose a threat to the system: they can execute double-spend attacks by mining their own branch until it eventually becomes the longest. But even less mining power might suffice since honest participants also discard the blocks of other honest miners if they are not on the same branch. This happens naturally in realistic networks due to propagation delays. If an attacker can induce and exploit communication delays, all discarded blocks may benefit the attacker; effectively increasing their strength [24].

In contrast, Tailstorm implements a *parallel* PoW consensus mechanism that largely avoids discarding blocks. For this, we closely follow the approach taken by Keller and Böhme at AFT '22 [29]. Their protocol, $\mathcal{B}_k$, confirms each block with k *votes*. $\mathcal{B}_k$ blocks do not require a PoW, only votes do. Notably, votes confirming the same block can be mined in parallel because they do not depend on each other. Discarding only occurs when there are more than k votes for the same block. Even then, individual discarded votes only account for $1/k$ of a block's PoW. As Keller and Böhme [29] argue, this makes consensus more robust against consensus layer attacks.

But it is futile to analyze the consensus without considering incentives. Ideally, rewards are distributed fairly, which means that a miner's expected reward is proportional to its hash rate. Arnosti and Weinberg [3] show that even small inequalities in reward allocation encourage substantial centralization of hash rate which ultimately poses a threat to consensus and the cryptocurrency itself [19].

Unfairness arises from natural network delays and dishonest behavior. Both of these factors affect Bitcoin. In latent networks, two blocks mined around the same time may refer to the same parent, even if both miners follow the longest chain rule. One of the blocks will be discarded, the other rewarded. The stronger miner has more hash rate to support their own block, hence the weaker miner is worse off. Network-level attackers can additionally exploit latency to manipulate impartial participants in their favor. But even without delays,

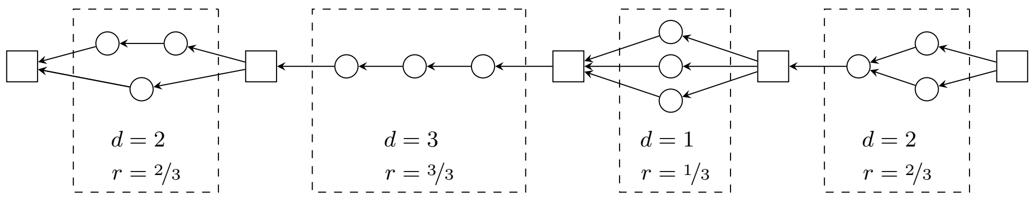


Figure 1 Example of a Tailstorm blockchain with $k = 3$ subblocks per summary. Squares are summary blocks, circles are subblocks, arrows indicate hash-references. Dashed rectangles mark subblock trees with depth d and discounted subblock reward r. Reduced depth implies lower rewards.

Bitcoin miners with more than one third of the hash rate can reap an unfair share of rewards by temporally withholding blocks instead of acting honestly [16, 21, 40, 47]. As we will demonstrate, $\mathcal{B}_k$ suffers from the same problem, partly due to its leader election mechanism.

Tailstorm addresses the unfairness problem through an innovative reward scheme that punishes withholding. The Tailstorm blockchain consists of *subblocks* and *summary* blocks. Similar to $\mathcal{B}_k$, summaries do not require a PoW, but subblocks do. Assembling a new summary requires k subblocks that confirm the same parent summary. To preserve the security properties of $\mathcal{B}_k$, subblocks confirming the same summary are conflict-free, and hence can be mined in parallel. Taking inspiration from Bobtail [9], subblocks *optionally* refer to another subblock instead of a summary, and hence form a tree. With Tailstorm, we propose to discount rewards based on the depth of this tree, as depicted in Figure 1. Mining subblocks in private causes branching of the tree, reduces its depth, and ultimately leads to lower rewards.

We support our claims with a comprehensive analysis and make the following contributions.

1. We formulate a generic system model for PoW cryptocurrencies. The models abstracts PoW and communication, defines a joint set of assumptions, and enables valid comparisons between different consensus protocols and incentive mechanisms.
2. We specify Bitcoin, $\mathcal{B}_k$, and Tailstorm in the joint model. To isolate the effect of Tailstorm's discount reward scheme and $\mathcal{B}_k$'s leader election mechanism, we additionally specify a hybrid protocol, TS/const, modelling Tailstorm without discounting and $\mathcal{B}_k$ without leader election.
3. We provide an upper bound for the orphan rate of Tailstorm in honest networks. Compared to Bitcoin [43], Tailstorm creates less orphans and hence presents less opportunity for unfairness.
4. We implement the system model as a simulator and show that Tailstorm is more fair than Bitcoin in honest but realistic networks with propagation delays. We confirm Bitcoin's inherent tradeoff: while short block intervals are desirable for fast confirmations and a less volatile stream of rewards, they also bias rewards in favour of strong miners. In Tailstorm, these concerns are largely separated by configuring long *summary* block intervals for fairness and short *subblock* intervals for fast confirmations and frequent rewards.
5. We evaluate multiple hard-coded attack strategies against the specified protocols, finding that attacks which are profitable against $\mathcal{B}_k$ are less profitable against Tailstorm, with the TS/const protocol lying in between.
6. We follow Hou et al. [26] and search optimal attack strategies using reinforcement learning. Our search reproduces optimal strategies against Bitcoin [21, 47] and generally matches or outperforms the hard-coded strategies. The regularity of our results indicates that we indeed found near-optimal strategies against all protocols and enables the conclusion that Tailstorm is less susceptible to incentive layer attacks than the other protocols.

7. We describe the Tailstorm application layer, which implements a cryptocurrency on top of the proposed consensus protocol. It preserves Bitcoin's transaction logic while enabling faster confirmations. Transactions are stored in subblocks in much the same way that the Storm protocol [4] stores transactions in delta blocks.
8. Lastly, we implement a prototype of Tailstorm which is ready for testnet deployments and make the code available online [23].

We structure the paper in order of our contributions. Section 2 defines the system model and Section 3 presents the specification of Tailstorm. In Section 4, we evaluate the protocols in an honest network with propagation delays. In Section 5, we evaluate hard-coded attack strategies, and in Section 6 we conduct the search for optimal policies with reinforcement learning. Section 7 presents the Tailstorm cryptocurrency and our prototype implementation. In Section 8, we discuss related work, limitations and future work. Section 9 concludes.

2 System Model

Recall the layered view on PoW cryptocurrencies presented in the introduction: The PoW layer moderates access using a mining puzzle, the consensus layer establishes a specific block ordering, the application layer writes cryptocurrency transactions into the blocks, and the incentive layer mints new cryptocurrency for successful miners. We now present a system model that abstracts PoW and communication to enable concise specification of the consensus layer. Application and incentives are considered in later sections.

In practice, PoW consensus protocols are executed as distributed systems, where independent nodes communicate over a P2P network. Messages exchanged between the nodes may be subject to natural or potentially malicious delays. To facilitate specification, we abstract this distributed system and model it algorithmically. We define a virtual environment that emulates distributed protocol execution in a single thread of computation. Within the virtual environment, nodes are represented as numbers, and blocks are represented as vertices in a directed acyclic graph (DAG). Mining is simulated as a loop with random delays, while communication is modeled by restricting the visibility of blocks to a subset of the nodes. The behaviour of nodes is defined by functions, which can be customized to model different protocols.

We define the virtual environment in Algorithm 1. The environment maintains a DAG where each *vertex* represents one block. Each block b has an associated list of parent blocks that constitute the outgoing *edges* in the DAG. We denote this list `parents`(b). In practice, edges arise from hash references pointing to other blocks in the blockchain. We say that block b is a *descendant* of block b', if b' is either a parent of b, or is connected transitively by the `parents` relationship. In this case, we say b' is an *ancestor* of b. Each block has *properties* which are assigned as the protocol unfolds. For example, the virtual environment uses the Boolean property `pow`(b) to track whether block b has a PoW or not.

We label the participating nodes with integers ranging from 1 to n. For each node, the virtual environment maintains a local view of the DAG and a preferred tip of the chain. In Lines 10 and 21, we restrict the local view of node i to blocks where `visibility`(b, i) was set to true. Initially, local views are empty and new blocks are not visible to any node. We denote as `tip`(i) the preferred *tip* of node i, the block to which a new block from i will point. We describe the behaviour of nodes as pure functions. These functions are called by the environment to obtain instructions from the node which the environment then interprets according to our assumptions. This makes all modifications of the DAG and all

Algorithm 1 Virtual Environment.

```
$t \leftarrow$ Root();                                   // Root: protocol's genesis block template
$b \leftarrow$ block obtained from appending template $t$ to the block DAG;   // reification
for $i = 1, \dots, n$ do                                 // $n$: number of nodes
    tip($i$) $\leftarrow b$;                             // tip($i$): state of node $i$
    visibility($b, i$) $\leftarrow$ true;                // visibility($b, i$): whether node $i$ sees block $b$

while true do                                            // concurrent PoW loop
    sample delay $d_{\text{pow}} \sim$ Expon(rate $\lambda$);   // $\lambda$: PoW rate
    sample node $i \sim$ Discrete($1, \dots, n$ with weights $\kappa_1, \dots, \kappa_n$); // $\kappa_1, \dots \kappa_n$: relative hash rates
    wait for $d_{\text{pow}}$ seconds, handling other tasks concurrently;
    with local view of node $i$ do                       // partial and immutable view on block DAG
        $tmpl \leftarrow$ Extend(tip($i$));              // Extend: protocol's mining rule
    $b \leftarrow$ block obtained from appending template $tmpl$ to the block DAG;   // reification
    pow($b'$) $\leftarrow$ true;                         // mark block as mined
    visibility($b, \cdot$) $\leftarrow$ false;           // block not yet propagated
    if Validate($b$) then Deliver($b, i$);               // Validate: protocol's chain structure

function Deliver(block $b$, node $i$) is                 // Turn block visible for node ...
    for $p \in$ parents($b$) do                          // ... in topological order ...
        wait until visibility($p, i$) is true;
    if not visibility($b, i$) then                       // ... and at most once.
        visibility($b, i$) $\leftarrow$ true;
        with local view of node $i$ do                   // partial and immutable view on block DAG
            (tip($i$), $share$, $append$) $\leftarrow$ Update(tip($i$), $b$); // Update: protocol's state update rule
        for $b' \in share$ and $j = 1, \dots, n$, $j \neq i$ do   // handle block broadcast requests
            pick message delay $d_{\text{net}}$ according to network assumptions;
            Deliver($b', i$) in $d_{\text{net}}$ seconds;
        for $tmpl \in append$ do                         // handle requests to append blocks without PoW
            $b' \leftarrow$ block obtained from appending template $tmpl$ to the block DAG; // reification
            pow($b'$) $\leftarrow$ false;                // block has no PoW
            visibility($b', \cdot$) $\leftarrow$ false;
            if Validate($b'$) then Deliver($b', i$);     // Validate: protocol's chain structure
```

communication explicit in Algorithm 1. In particular, nodes do not directly append blocks to the DAG; they return block templates, which the environment then reifies by appending a new block to the DAG.

A protocol is fully specified through four functions: `Root` and `Validate` define the structure of the blockchain, while `Update` and `Extend` define the behavior of honest nodes.

The `Root` function takes no argument and returns a single block, which we call *genesis*. Initially, the genesis is the only block in the DAG. `Validate` takes a block as argument and returns `true` if the block is valid and `false` otherwise. E. g., Bitcoin's `Validate` checks the `pow` property and that there is exactly one parent. The environment enforces block validity during the reification of blocks, while deployed protocols would reject invalid blocks in the communication layer. The genesis is not subject to the validity rule.

The `Update` function specifies how nodes react to newly visible blocks, after they are mined locally with PoW, appended locally without PoW, or received from the network. The function takes two arguments: the node's currently preferred tip and the new block. The function returns the new preferred tip, a list of blocks the node intends to *share* with other nodes, and a list of block templates it wants to *append* to the chain *without* PoW. On the other hand, the function `Extend` defines how nodes grow the chain *with* PoW. It takes a single argument, a node's currently preferred tip, and returns a template for the block that the node intends to mine.

Algorithm 2 Tailstorm: Chain Structure.

```
function Root() is
    return block template $b$ with summary($b$) = true and height($b$) = depth($b$) = 0;

function Validate(block $b$) is
    if summary($b$) then
        $p \leftarrow$ last summary before $b$;
        $S \leftarrow$ subblocks between $b$ and $p$;
        return $(|S| = k) \wedge$ (depth($b$) $= 0) \wedge$ (height($b$) = height($p$) $+ 1)$;
    else                                                        // $b$ is subblock
        $p \leftarrow$ first parent of $b$;
        return
          pow($b$) $\wedge$ (|parents($b$)| $= 1) \wedge$ (depth($b$) = depth($p$) $+ 1) \wedge$ (height($b$) = height($p$));
```

We follow related work [51, 43, 15, 37, 29] and model the mining process in continuous time. The virtual environment generates independent mining delays from the exponential distribution $\texttt{Expon}(\lambda)$, with rate λ measured in expected number of proofs-of-work per second. Accordingly, the expected value of the distribution, $1/\lambda$, is called the *mining interval* and is measured in seconds. After each mining delay, the environment randomly selects a successful miner, obtains a block template from `Extend`, and reifies the block by appending it to the DAG. We support arbitrary hash rate distributions among the nodes by setting the weights $\kappa_1, \ldots, \kappa_n$ in Line 8 accordingly.

The `Deliver` function captures the process of making blocks visible to nodes. The specified protocols have in common that block validation requires knowledge of all referenced blocks. We avoid a lot of boilerplate code in the specification, by ensuring that parent blocks are delivered before their children. Upon delivery, the virtual environment first invokes the `Update` function to obtain the node's new preferred tip, a list of blocks the node wants to share, and a list of block templates the node intends to append without PoW. It then handles the node's requests to share and append. Communication is modelled through delayed delivery, while appends happen immediately.

3 The Tailstorm Protocol

This section specifies the Tailstorm consensus protocol and reward mechanism using the algorithmic model described in Section 2. The specification serves as the basis for our theoretical analyses, network simulations, and attack search in subsequent sections. We first describe Tailstorm's chain structure in Section 3.1. We then specify the behaviour of honest nodes in Section 3.2. As a point of reference, we also specify Bitcoin and $\mathcal{B}_k$ protocols in the extended version of the paper. In this section, we assume that the application layer implements a cryptocurrency which we can use to pay rewards. We defer the description of Tailstorm's application layer to Section 7. Throughout this section, we focus on honest miners who follow the protocol as intended. Later sections will consider dishonest behaviour.

3.1 Chain Structure

Algorithm 2 defines Tailstorm's chain structure. Each block is either a summary or a subblock. Subblocks must have PoW and they must have exactly one parent. Summaries do not require PoW and reference k subblocks, each confirming the same ancestor summary.

Each block b has two integer properties, $\texttt{height}(b)$ and $\texttt{depth}(b)$. The genesis has height and depth zero. Subblocks inherit the height of their parent and they increment the depth by one. Summaries increment the height and reset the depth to zero. Figure 1 in Section 1

Algorithm 3 Tailstorm: Node.

```
function Preference(summary $s$, summary $b$) is
    $hs$, $hb \leftarrow$ height($s$), height($b$);
    $ns$, $nb \leftarrow$ size of subblock tree confirming $s$ and $b$;
    $rs$, $rb \leftarrow$ reward obtained from $(s, b)$;
    return $(hb > hs) \lor (hb = hs \land nb > ns) \lor (hb = hs \land nb = ns \land rb > rs)$;

function Extend(tip $t$) is                                  // $t$: currently preferred block
    if $t$ has children then $p \leftarrow$ maximum-depth subblock after $t$ else $p \leftarrow t$;
    return subblock template $b$ with
      parents($b$) = $[p]$, height($b$) = height($p$), and depth($b$) = depth($p$) + 1;

function Update(tip $t$, block $b$) is          // $t$: currently preferred block // $b$: new block
    $pref$, $share$, $append \leftarrow t, [b], []$; // new preferred tip, blocks to broadcast, templates to append
    if summary($b$) then
        if Preference($pref$, $b$) then $pref \leftarrow b$;
    else                                                               // $b$ is subblock
        $p \leftarrow$ last summary before $b$;
        if Preference($pref$, $p$) then $pref \leftarrow p$;
        $S \leftarrow$ all subblocks in subblock tree confirming $p$;
        if $|S| \geq k$ then
            $L \leftarrow$ result of Algorithm 4: Subblock Selection;
            create summary template $s$ with parents($s$) = $L$;
            $append \leftarrow [s]$;
    return $pref$, $share$, $append$;
```

illustrates a valid Tailstorm chain for $k = 3$. Note that the subblocks confirming the same summary form a tree and that the `depth` property tracks the depth of this tree. The `height` property on the other hand counts the number of trees that have been summarized.

To incentivize participation, Tailstorm allocates rewards to the miners of subblocks. The reward size is proportional to the depth of the subblock tree: let b be a summary block, and S be the set of subblocks in the corresponding subblock tree. Then all subblocks in S are allocated the same reward

$$\texttt{discount}(b) = \frac{c}{k} \cdot \max_{x \in S}(\texttt{depth}(x)), \qquad (1)$$

where c represents a tunable upper limit on the subblock reward. In Figure 1 we show the rewards for $c = 1$. Note that the reward scheme punishes non-linearities in the blockchain, and this punishment affects all included subblocks equally.

3.2 Honest Nodes

Algorithm 3 specifies the behaviour of honest nodes. The algorithm revolves around a preference order (ln. 1-5) that ranks summaries first by height, then by number of confirming subblocks, and finally by potential personal reward for the individual node. Nodes set the highest ranked summary as their preferred summary (ln. 13+16) and they mine subblocks (ln. 6-9) that confirm their preferred summary. To maximize the depth of the subblock tree, nodes append their subblocks to the longest existing branch.

Whenever nodes learn about a new block (ln. 10-22), they share it with the other nodes and they update their preference. As soon as there are k subblocks (ln. 18) confirming the preferred summary, nodes assemble the next summary. When there are more than k subblock candidates for the next summary, nodes choose the subblocks to maximize their own rewards. We present a greedy algorithm for subblock selection in Algorithm 4. Note that nodes may

Algorithm 4 Tailstorm: Subblock Selection.

```
1  $R \leftarrow \emptyset$;                                                  // selected subblocks
2  while $|R| < k$ do                                         // select one subblock per iteration
3      $C \leftarrow S \setminus R$;                                       // $S$: candidate subblocks
4      for $x \in C$ do
5          $B_x \leftarrow x$ and all ancestors of $x$ in $S$;
6          $B'_x \leftarrow B_x \setminus R$;                                // newly referenced blocks
7          $r_x \leftarrow$ number of node's own subblocks in $B'_x$;                    // reward
8      $C \leftarrow \{x \in C : |R| + |B'_x| \leq k\}$;                        // enforce tree size $k$
9      $y \leftarrow \arg\max_{x \in C} r_x$;                  // select candidate with highest reward
10     $R \leftarrow R \cup B'_y$;                                   // track referenced subblocks
11 return leaves in subblock tree $R$;                                     // invariant: $|R| = k$
```

choose different subblocks and thereby create conflicting summaries. Such conflicts create temporary forks of the blockchain which are resolved quickly according to the preference order described above. We analyze the implications on fairness in Section 4.

3.3 Difficulty Adjustment

A major goal for most blockchains, including Tailstorm, is for the blockchain itself to grow at a constant rate so as to maintain constant transactional throughput. However, in any deployed blockchain, the puzzle solving rate λ changes over time because nodes may come and go or may add or remove mining hardware. The changes in solving rate lead to changes of the growth rate. Adjusting for the fluctuations requires feedback from the consensus to the PoW layer. Typically, blockchains adjust the puzzle solving *difficulty* depending on the observed chain growth using a dynamic *difficulty adjustment algorithm* (DAA).

There exists a rich body of prior work concerning DAA design and analysis [8, 18, 25, 27, 35], but a deep investigation of ideal DAAs for the Tailstorm protocol is beyond the scope this paper. We note, however, that existing DAAs for Bitcoin can be adapted to Tailstorm by counting the number of subblocks where Bitcoin DAAs use the length of the blockchain. For any Tailstorm block b (summary or subblock), we define

$$\texttt{progress}(b) = k \cdot \texttt{height}(b) + \texttt{depth}(b)\,, \tag{2}$$

which counts the number of PoWs included in the chain. Any Tailstorm DAA should adjust the puzzle solving difficulty such that `progress` grows at a constant rate.

3.4 Protocol Variant With Constant Rewards

Tailstorm discounts rewards proportional to the depth of the subblock tree. We see the discounting mechanism as a core contribution of this paper. To isolate the effect of discounting, we introduce a protocol variant without discounting, which we call TS/const. While Tailstorm pays out *at most c* units of reward per subblock, the TS/const protocol pays out *exactly c* units of reward per subblock. In all other aspects, TS/const is identical to Tailstorm.

Note, that the TS/const protocol does not use the subblock tree structure, neither for consensus nor for incentives. In that regard, TS/const resembles the parallel PoW protocol $\mathcal{B}_k$ [29] where all subblocks refer to a summary, never another subblock. The only difference with $\mathcal{B}_k$ is that TS/const does not implement leader election: while $\mathcal{B}_k$ restricts the creation of the next summary to the miner of the subblock with the smallest hash, any TS/const node may (re-)create valid summaries locally.

4 Fairness Under Protocol Compliance

A fair PoW protocol rewards miners in proportion to the amount of work they do. Disproportionate or inconsistent allocation discriminates against weak miners and encourages the formation of pools and centralization. In this section, we explore the causes of unfairness in Tailstorm under the assumption that all miners are honest. We measure work in number of hashes evaluated. The system is fair if rewards are proportional to the miners' hash rates.

The root cause of unfairness is the inherent asymmetry in reward loss when multiple PoW solutions are discovered in short order. In Tailstorm, if more than k subblocks are produced, all having the same height, then all but k of them will be discarded. The discarded blocks are commonly called *orphans* and receive no rewards. Typically, miners do not orphan their own blocks. Since miners with a high hash rate are more often able to choose which blocks will be orphaned, miners with a relatively low hash rate lose a disproportionate amount of rewards. In Bitcoin, this effect is amplified because orphaning can occur for each block (set $k = 1$ in the above argument).

4.1 Analytical Orphan Rate Analysis

In this section, we develop an analytical model for the orphan rate and use it to compare the fairness of Bitcoin and Tailstorm. Let B be a random variable denoting the number of subblocks orphaned during the production of a summary block in Tailstorm or the number of orphans per block in Bitcoin. The orphan rate is given by $\rho = E[B]/k$, which represents the expected number of PoW solutions orphaned for every PoW solution confirmed. In the remainder of this section, we will derive a bound on ρ.

Following Rizun's orphan rate analysis for Bitcoin [43], we model block propagation delays as

$$\tau(\tau_0, z, Q) = \tau_0 + zQ, \tag{3}$$

where τ_0 represents network latency (seconds), z represents bandwidth (bytes per second), and Q represents block size (bytes). In order to adapt this expression for Tailstorm, we assume that the transactions included in a summary block are spread evenly across the k subblocks, so the subblock size is Q/k. The expected summary block interval is T and the subblock mining rate is $\lambda = k/T$.

▶ **Theorem 1.** *For network parameters τ_0, z, and Q, summary block interval T, and subblock count k, Tailstorm's expected orphan rate ρ is bounded from above by:*

$$\rho(\tau_0, z, Q, T) \leq \frac{\tau_0}{T} + \frac{zQ}{kT}.$$

Proof. Assume there are $k + X$ subblocks, all pointing to the same summary block. Then, k of these subblocks will be included in the next summary and X will be orphaned. Recall from Algorithm 3 that honest miners proceed to the next summary as soon as they learn of k subblocks. So, if X subblocks are orphaned, they must have been mined before the kth subblock has propagated to all nodes. Additionally, subblocks k through $k + X$ must originate from different miners, since no miner orphans its own subblock.

Let Y_t be a random variable representing the number of subblocks generated during an arbitrary time interval $t > 0$ under the assumption, introduced in Section 2, that mining intervals are exponentially distributed with rate $\lambda = k/T$. A standard result from the theory of Poisson processes shows that $Y_T \sim \texttt{Poisson}(k)$ [46, ch. 5]. The same theory shows that

Table 1 Upper bounds on orphan rate obtained from Theorem 1 assuming various combinations of k and expected block (Bitcoin) or summary block (Tailstorm) intervals T.

	Bitcoin	Tailstorm		
block interval	$k = 1$	$k = 5$	$k = 10$	$k = 15$
$T =$ 75 seconds	10.08 %	7.35 %	7.01 %	6.89 %
$T =$ 150 seconds	5.04 %	3.67 %	3.50 %	3.45 %
$T =$ 300 seconds	2.52 %	1.84 %	1.75 %	1.72 %
$T =$ 600 seconds	1.26 %	0.92 %	0.88 %	0.86 %

$Y_t \sim \texttt{Poisson}(t\frac{k}{T})$ for arbitrary $t > 0$. According to our network model, the kth subblock (as well as the $k-1$ before) is fully propagated after $\tau(\tau_0, z, Q/k)$ seconds. Thus, X is equal to $Y_{\tau(\tau_0,z,Q/k)}$ in distribution. $E[X]$ bounds the expected number of subblocks orphaned per summary block. The orphan rate, that is subblocks orphaned per subblocks included, is bounded by

$$\rho(\tau_0, z, Q, T) \leq \frac{E[X]}{k} = \frac{\tau(\tau_0, z, Q/k)\frac{k}{T}}{k} = \frac{\tau_0}{T} + \frac{zQ}{kT}. \tag{4}$$

◀

Table 1 shows how the upper bound varies depending on the expected summary block interval T and the subblock count k. The orphan rate for Bitcoin [43] appears in the column for $k = 1$. In this analysis we assume that $Q = 32\,\text{MB}$, $z = 100\,\text{MB/s}$, and $\tau_0 = 5\,\text{s}$. Generally, we can see that the orphan rate decreases with both expected summary block interval T and k. This suggests that, when summary block intervals are constant across protocols, Tailstorm with high k increases fairness over Bitcoin and Tailstorm with lower k.

4.2 Measuring Fairness in Simulation

Section 4.1 provides an analytical bound for Tailstorm's orphan rate. This bound is only tight if all subblocks originate from different miners, which is unlikely to happen in practice. To obtain a more realistic analysis, albeit in a more specific setting, we now consider strong miners who are likely to produce multiple blocks within a single propagation delay. Since the mathematical analysis for this scenario is complex, we rely on simulation instead. Additionally, instead of using the orphan rate metric, itself only a proxy for fairness, we directly measure a miner's deviation from their fair share of rewards.

We implement the virtual environment as described in Section 2, Algorithm 1 and measure how the choice of protocol – Bitcoin and Tailstorm under various parameterizations – affects rewards. The network is configured with one *weak* and one *strong* miner operating with 1 % and 99 % of the total hash rate, respectively. All messages are delayed for 6 seconds.

We simulate one million PoW solutions per configuration. We split the simulation into *independent observations*, each representing one day of protocol execution. Specifically, with T denoting the expected summary block interval, we terminate individual executions when $\lfloor 24 \times 3600 \times k/T \rfloor$ PoWs are mined. For each execution, we identify the longest chain of blocks and calculate the accumulated rewards for both miners. Since each observation covers one day of mining, the variance represents daily volatility of rewards.

A miner's fair share of reward equals its relative hash rate. To measure fairness, we compare actual rewards to the fair share. Figure 2 shows the weak miner's relative reward as a percentage of its fair share. The leftmost facet shows the Bitcoin protocol for expected

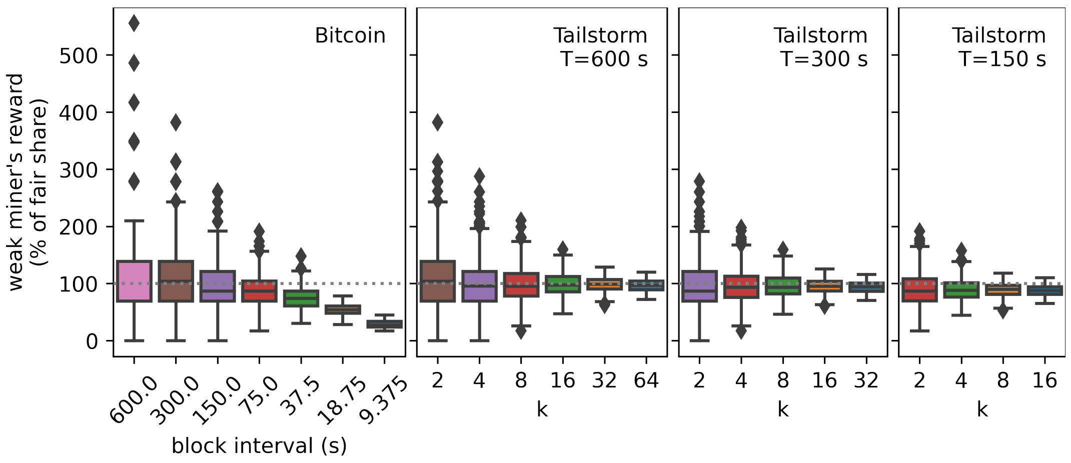


Figure 2 Observed fairness for different configurations of Bitcoin (leftmost facet) and Tailstorm (right facets) with expected summary interval T ranging from 600 to 150 seconds. Colors represent expected subblock mining intervals.

block intervals ranging from roughly 9 seconds to 600 seconds. The remaining three facets show Tailstorm with *summary* block intervals T of 600, 300, and 150 seconds for k ranging from 2 to 64. Like colors represent such configurations where *subblocks* are generated at the same frequency $\lambda = k/T$. We omit the configurations where the expected subblock interval T/k is lower than the network propagation delay of 6 seconds.

The Figure illustrates the tradeoff between reward fairness and volatility in Bitcoin and Tailstorm. In the leftmost facet, it shows how increasing the expected block interval improves fairness but also increases volatility in Bitcoin. In contrast, the right facets show how Tailstorm can be configured to independently control both reward fairness (by adjusting the summary block interval T) and volatility (by adjusting k). In particular, choosing higher k leads to more frequent rewards and lower volatility, while longer summary block intervals tend to increase fairness. The latter observation aligns with the results in Table 1, which also shows that fairness tends to increase with the summary block interval.

5 Attack Evaluation

We now extend the virtual environment defined by Algorithm 1 in Section 2 to model adversarial behavior. We then specify and evaluate several attack strategies against Tailstorm.

To account for the possibility of collusion among multiple dishonest parties, we pessimistically assume that they indeed collude. Therefore, we model a single but strong node deviating from the protocol. Throughout the remaining sections we refer to the single dishonest node as the *attacker* and to the honest nodes as *defenders*. The attacker *observes* the virtual environment's state and reacts to any updates accordingly. We describe attacks as *policies*, which determine the *action* to take based on the observed state.

5.1 Network

We deploy a virtual environment comprising n nodes. The first node represents the attacker and is assigned fraction α of the total hash rate. The remaining hash rate is distributed evenly among the $n-1$ defenders. Henceforth, we refer to α as the *attacker's strength*.

In Bitcoin, blocks form a linear chain. If there are two blocks with the same height, only one will be included in the blockchain long term. Since honest Bitcoin nodes prefer and mine on the block first received, nodes with better connectivity obtain an advantage. The same *block race* situation can arise in Tailstorm. Here, there can be only one *summary* at each height and the nodes' preference of summaries (Alg. 3) likewise depends on the order they are received. Attackers able to send blocks more quickly might be able to manipulate the defenders' preference to suit their needs.

We follow Eyal and Sirer [16] and Sapirshtein et al. [47] who model the *block race advantage* with a single network parameter γ, which defines the proportion of defenders, weighted by hash rate, that will opt for the attacker's block in the case of a block race. We pessimistically assume that the attacker receives all blocks without delay. Defender nodes can send blocks to one another with a negligible delay of ε. When the attacker sends a block to a defender, we introduce a random delay, which we draw independently from a uniform distribution on the interval $[0, \frac{n-2}{n-1}\frac{\varepsilon}{\gamma}]$.

Note that we chose these delays to enable comparison with related work [16, 47] on selfish mining against Bitcoin. Other constructions with different distributions are possible but redundant: $\gamma = 0$ implies an inferior network level attacker who wins *no* block races, $\gamma = 1$ models a superior attacker who wins *all* block races, and intermediate attackers can be modeled by choosing γ accordingly. The effect of non-negligible delays among the defenders was analyzed in Section 4.

▶ **Theorem 2.** *Let $\gamma \in [0, 1]$ be given. For any choice of $\varepsilon > 0$ and $n > \frac{1}{1-\gamma} + 1$, our definition of γ is equal to the attacker's block race advantage in [16, 47].*

Proof. Let A be the event that a randomly chosen honest miner accepts the attacker block over honest and A_i the event that a specific miner accepts the attacker block. By the law of total probability we have $P[A] = \frac{1}{n-1}\sum_{i=1}^{n-1} P[A_i]$. Note also that if we assume w.l.o.g. that A_{n-1} was the miner of the competing block, then $P[A_{n-1}] = 0$. Since defenders communicate with each other with delay ε while the attacker communicates with defenders with uniform random delay up to time $\frac{n-2}{n-1}\frac{\varepsilon}{\gamma}$, it must be the case that

$$P[A_i] = \varepsilon \left(\frac{n-2}{n-1}\frac{\varepsilon}{\gamma}\right)^{-1}, \quad \forall i < n-1, \tag{5}$$

provided that $\frac{n-2}{n-1}\frac{\varepsilon}{\gamma} > \varepsilon$. Choosing $n > \frac{1}{1-\gamma} + 1$ satisfies this constraint. Equality in the constraint is not valid because it forces $P[A_i] = 1$ for all γ. Therefore,

$$\begin{aligned} P[A] &= \frac{1}{n-1}\sum_{i=1}^{n-1} P[A_i] && (6)\\ &= \frac{1}{n-1}\sum_{i=1}^{n-2} P[A_i] && (7)\\ &= \frac{n-2}{n-1}\frac{\varepsilon}{\frac{n-2}{n-1}\frac{\varepsilon}{\gamma}} && (8)\\ &= \gamma. && (9) \end{aligned}$$

◀

Constraint $n > \frac{1}{1-\gamma} + 1$, from Theorem 2, is fundamental; if n is smaller, then it is not possible for an honest miner to communicate a block to another honest miner faster than the attacker can. The constraint implies that $\gamma < \frac{n-2}{n-1}$. Hence, we can model $\gamma = 1$ only in the

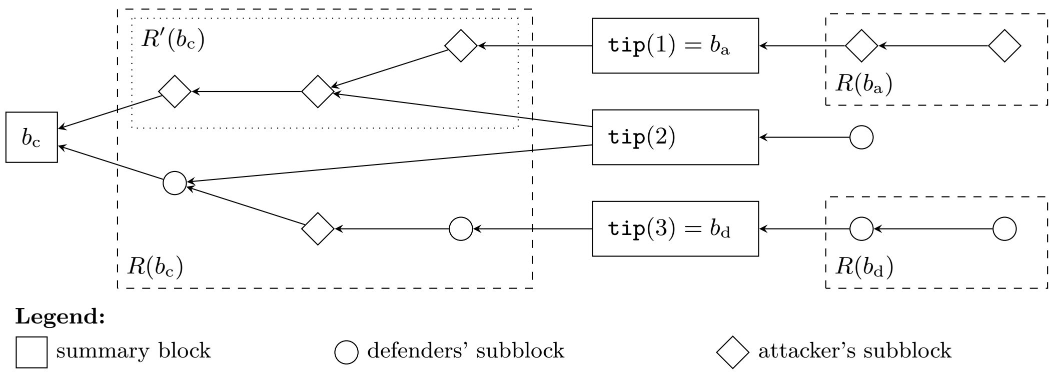


Figure 3 Notation in Section 5.2 Observation Space. This example uses three nodes and $k = 3$.

limit that the number of miners approaches infinity, which is not possible in practice. Other authors [16, 47] have directly considered $\gamma = 1$, but we feel that the restriction to $\gamma < 1$ is natural. Given that the miner of the defending block in a block race will never adopt the attacker's competing block, $\gamma = 1$ implies that individual miners have zero hash rate, which is not realistic.

5.2 Observation Space

The virtual environment maintains complex state. It tracks the full history of blocks, delayed messages, and the partial views of all nodes. Some of this information is not available to attackers in practice, other parts are not relevant. To enable concise policies, we follow Sapirshtein et al. [47] and restrict what the attacker can see.

Figure 3 illustrates our notation. At any given time, let $b_a = \mathtt{tip}(1)$ denote the attackers preferred summary block, and let b_d denote the best block (according to `Preference` in Alg. 3) among the preferred blocks of the defenders: $\mathtt{tip}(2), \ldots, \mathtt{tip}(n)$. We use b_c to refer to the best summary block among the common ancestors of b_a and b_d. In other words, b_c is the latest block that *all* nodes agree upon. For any summary block b, we use $R(b)$ to identify the subblocks which are both a descendant of b and have the same height as b. Note that Tailstorm's chain structure (Alg. 2) implies that $R(b)$ and b together span a tree. Section 3 defines the *depth* of a subblock to be its depth within this tree. We define $R'(b)$ as the largest subset of $R(b)$ such that all subblocks are mined by the attacker and $R'(b) \cup \{b\}$ is still connected. In the following, we use $|S|$ to refer to the cardinality of a set S and $\mathtt{depth}[S]$ to refer to the maximum depth among all blocks in S. The attacker observes

h_a: the attacker's height advantage, $\mathtt{height}(b_a) - \mathtt{height}(b_c)$,
h_d: the defenders' height advantage, $\mathtt{height}(b_d) - \mathtt{height}(b_c)$,
s_a: the attacker's inclusive subblock count, $|R(b_a)|$,
s'_a: the attacker's exclusive subblock count, $|R'(b_a)|$,
s_d: the defenders' subblock count, $|R(b_d)|$,
d_a: the attacker's inclusive depth, $\mathtt{depth}[R(b_a)]$,
d'_a: the attacker's exclusive depth, $\mathtt{depth}[R'(b_a)]$, and
d_d: the defenders' depth, $\mathtt{depth}[R(b_d)]$.

The example shown Figure 3 implies $(h_a, h_d, s_a, s'_a, s_d, d_a, d'_a, d_d) = (1, 1, 2, 2, 2, 2, 2, 2)$.

Algorithm 5 Tailstorm: Attacker.

```
function Update(tip $t$, block $b$) is            // $t$: currently preferred block // $b$: new block
    $pref, share, append \leftarrow t, [], []$;  // new preferred tip, blocks to broadcast, templates to append
    if $b$ is own summary block then $pref \leftarrow b$;
    $W \leftarrow$ set of withheld blocks (= blocks not shared before);
    $withhold, extend \leftarrow \mathcal{P}(h_a, h_d, s_a, s'_a, s_d, d_a, d'_a, d_d)$;            // $\mathcal{P}$: attack policy
    switch withhold do
        case Adopt do $pref \leftarrow b_d$            // $b_d$: defender's block, see Sec. 5.2 ;
        case Match do $share \leftarrow \{x \in W \mid \texttt{progress}(x) \leq \texttt{progress}(b_d)\}$ ;
        case Override do $share \leftarrow \{x \in W \mid \texttt{progress}(x) \leq \texttt{progress}(b_d) + 1\}$ ;
        case Wait do nothing;
    switch extend do
        case Inclusive do
            if $|R(pref)| \geq k$ then            // $R(b)$: full subblock tree confirming $b$, see Sec. 5.2
                $append \leftarrow$ summary template using $R(pref)$;
        case Exlusive do
            if $|R'(pref)| \geq k$ then            // $R'(b)$: partial subblock tree confirming $b$, see Sec. 5.2
                $append \leftarrow$ summary template using $R'(pref)$;
    return $pref, share, append$;
```

5.3 Action Space

The virtual environment (Alg. 1) calls a node's `Update` function whenever this node learns about a new block: after it was mined locally with PoW, appended locally without PoW, or received from the network. We implement the attacker's *dishonest* `Update` function in Algorithm 5. We allow for generic attacks by letting the attacker choose from a set of potentially dishonest actions. We assume that there is a policy $\mathcal{P}$ which maps observations (see Sect. 5.2) to action tuples of the form (*withhold*, *extend*). The *withhold* action type controls the preferred tip of the chain and the withholding of blocks. We hereby follow closely the actions used by Sapirshtein et al. [47] for selfish mining against Bitcoin [16].

`Wait` Continue mining on b_a and withhold new blocks.
`Match` Release just enough blocks to induce a block race between b_d and an attacker block.
`Override` Release just enough blocks to make the defenders discard their block b_d.
`Adopt` Abort attack, prefer defenders' block b_d, and discard b_a.

Recall from Section 2 that "blocks" refers to vertices in the block DAG. This makes the *withhold* action protocol-agnostic. For Tailstorm, blocks can be subblocks or summaries. Note that while `Adopt` and `Wait` are always feasible, `Match` and `Override` are not. If the attacker's chain is too short, the defenders will not consider adopting it. In such cases, `Match` and `Override`, as implemented in Algorithm 5, fall back to releasing all withheld blocks.

The second action type, *extend*, is specific to Tailstorm. It controls how the attacker assembles new summary blocks, more specifically, which subblocks it considers for selection with Algorithm 4.

`Inclusive` Use all available subblocks to create new summaries.
`Exclusive` Use only subblocks which were mined by the attacker.

Note that the attacker never delays the next summary block longer than necessary. As soon as there are enough subblocks for an inclusive or exclusive summary (according to the chosen action), this summary will be created.

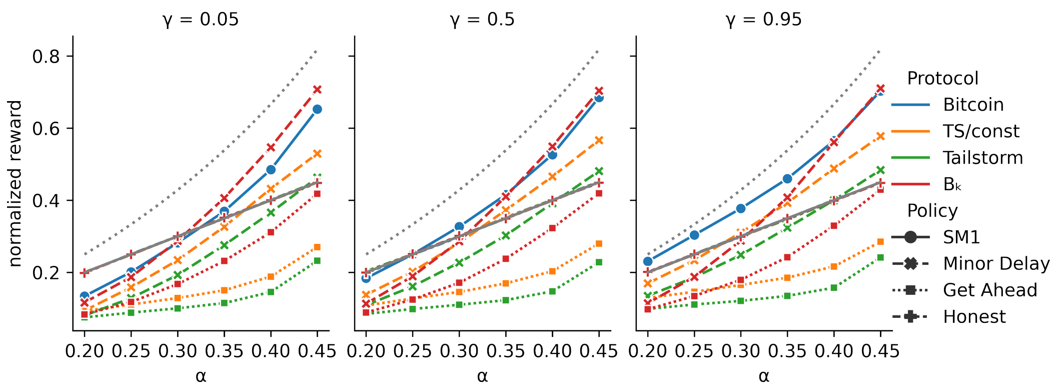


Figure 4 Observed normalized reward as a function of attacker strength α for different protocols (color) and reference policies (line style and markers). The three facets represent the network assumptions $\gamma \in \{5, 50, 95\}$ %. We set protocol parameter $k = 8$ where applicable. TS/const is Tailstorm without reward discounting. We add a gray reference line for the fair reward α (overlaps with honest policy) and a gray dotted curve for the known upper bound $\alpha/(1-\alpha)$.

5.4 Reference Policies

We evaluate the following policies. In each, the attacker assembles `Inclusive` summaries.

Honest Emulate Algorithm 3: adopt the longest chain and release all blocks:
`Adopt` if $h_\mathrm{d} > h_\mathrm{a}$. Otherwise `Override`.

Get Ahead Withhold own subblocks, release own summaries:
`Adopt` if $h_\mathrm{d} > h_\mathrm{a}$. `Override` if $h_\mathrm{d} < h_\mathrm{a}$. Otherwise `Wait`.

Minor Delay Withhold own subblocks, override defender summaries as they come out:
`Adopt` if $h_\mathrm{d} > h_\mathrm{a}$. `Wait` if $h_\mathrm{d} = h_\mathrm{c}$. Otherwise `Override`.

To evaluate the policies above, we reuse the simulator from Section 4.2, configuring the network as described in Sections 5.1 to 5.3. We set block race advantage $\gamma \in \{5, 50, 95\}$ % and allow the attacker strength α to range from 20 % to 45 %. We run one hundred simulations per configuration, and stop each as soon as the DAG contains 2048 blocks. At the end of each simulation we select the longest chain of blocks and calculate its rewards. We configure Tailstorm's discount reward scheme, as defined in Section 3.1, with $c = 1$ such that at most one unit of reward is minted per unit of chain progress.

Some policies cause more orphans than others. Assuming effective difficulty adjustment according to Section 3.3, the amount of simulated time depends on the attacker's behaviour. The attacker's mining cost is proportional to the time spent on the attack. To facilitate comparison across policies, we normalize rewards with respect to simulated time. Formally, we define the *normalized reward* as the attacker's reward up to block b divided by $\texttt{progress}(b)$. Note that relative reward, a metric commonly used for Bitcoin [55, 16, 47, 54], is not sufficient to account for Tailstorm's reward discounting.

Figure 4 reports the average normalized reward (across 100 simulations) on the y-axis for the varying α on the x-axis and with γ varying by facet. The green curves represent the different reference policies against Tailstorm. We also evaluate the reference policies against the TS/const protocol as defined in Section 3.4 (orange curves) and against $\mathcal{B}_k$ (red curves). In addition, we evaluate the SM1 policy against Bitcoin (blue curve) as described by Sapirshtein et al. [47]. For orientation we include their upper bound $\alpha/(1-\alpha)$ as a gray dotted curve and add a solid gray line for α, the expected reward of honest behaviour.

The figure supports multiple conclusions. First, the curves for the *Honest* policy coinciding with the line for α indicates that the policy indeed replicates honest behavior in all evaluated protocols. Second, comparing the measurements for Tailstorm with the ones for TS/const shows clearly that the discounting of rewards reduces efficacy of all dishonest reference policies. Third, *Minor Delay* is generally the most effective reference policy. Forth, *Minor Delay* produces less reward in Tailstorm than SM1 does in Bitcoin. Fifth, for low γ, *Minor Delay* is more profitable against $\mathcal{B}_k$ than SM1 is against Bitcoin.

Note however, that *Minor Delay* might not be the best strategy against Tailstorm, just like SM1 is not always optimal for Bitcoin [47]. This motivates to search for optimal policies.

6 Attack Search

Previous research has identified optimal attacks against Bitcoin and Ethereum through the use of Markov Decision Processes (MDP) and exhaustive search [55, 21, 47]. However, for more complex protocols, the state space of MDPs can become prohibitively large, and exhaustive search becomes impractical [26, 29, 54]. As a result, many authors in the past have resorted to evaluating hard-coded attacks instead of searching for optimal policies [9, 17, 29, 34, 13].

We adopt the approach introduced by Hou et al. [26] and employ reinforcement learning (RL) to search for attacks. The observation and action spaces described in Section 5 readily define a partially observable MDP. To search for optimal attacks, we replace the hard-coded policy P with an RL agent that learns to select actions based on past observations.

We utilize off-the-shelf RL tooling by first exposing our simulator and attack space as an OpenAI Gym [10]. Next, we deploy Proximal Policy Optimization (PPO) [48] as our agent. To support reproduction and future research, we release the Gym as Python package [28] and include all training scripts in our open source repository [30].

The modularity of our simulator allows us to apply the same RL pipeline to different protocols. We insert the protocol specifications along with their associated observation and action spaces, while reusing the virtual environment and network assumptions across protocols. For Bitcoin, we adopt the attack space defined by Sapirshtein et al. [47]. Details about the attack spaces for Bitcoin and $\mathcal{B}_k$ are provided in the extended version of this paper.

We use Bitcoin as a reference protocol to measure the completeness of our approach. Previous research by Sapirshtein et al. [47] has yielded the optimal policy. As we will demonstrate, our search mechanism reproduces these results closely.

We search for optimal policies for all possible combinations of attacker strength $\alpha \in \{20, 25, 30, \ldots 45\}\,\%$, and block race advantage $\gamma \in \{5, 50, 95\}\,\%$. We evaluate four protocols: Bitcoin, $\mathcal{B}_k$, Tailstorm, and TS/const with $k = 8$ where applicable. In total, this amounts to $7 \cdot 3 \cdot 4 = 84$ different learning problems.

For each learning problem, we conduct multiple training runs with varying hyperparameters. For the objective function we choose *normalized reward*, as evaluated for the reference policies in Figure 4. From the training we obtain 1292 policies.

We select the best trained policy for each problem by simulating 100 independent protocol executions for each and proceeding as follows. As in Section 5.4, we stop individual executions as soon as there are 2048 blocks. We then select the longest chain of blocks and observe the attacker's normalized reward. This reward is averaged over the 100 observations per policy to determine the policy with the highest average reward. For reference, we apply the same filtering method to select the best policy among the reference policies from Section 5.4.

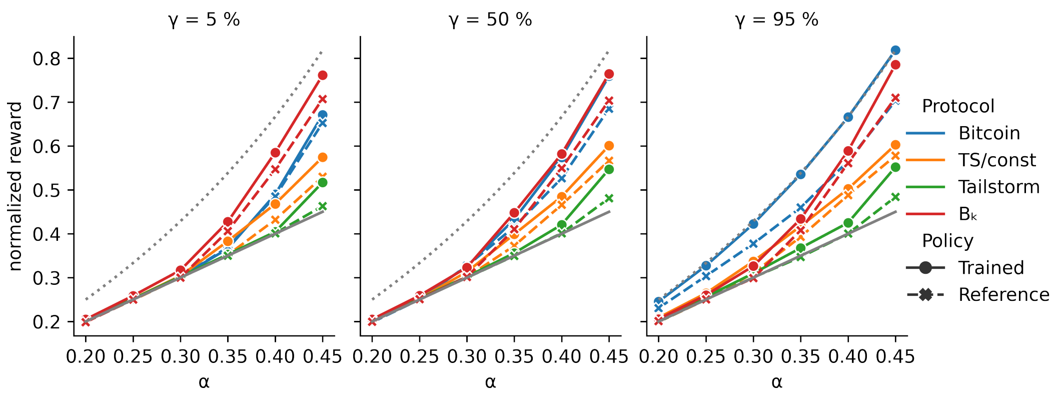


Figure 5 Observed normalized reward of the *best known policy* with the axes set up like in Figure 4. Solid curves with •-markers represent trained reinforcement learning models. Dashed curves with ×-markers represent the reference policies defined in Section 5.4.

Figure 5 shows the performance of the selected policies, with γ varying by facet, α on the x-axis, and average normalized reward on the y-axis. The solid colored curves represent the best trained policies, while the dashed colored curves represent the best reference policies. As in Figure 4, we include gray reference curves for Bitcoin's known upper bound $\alpha/(1-\alpha)$ (dotted) and fair share α (solid). Note that for $\gamma = 50\,\%$, Nakamato and $\mathcal{B}_k$ visually overlap.

In comparing the performance of different protocols, we say that protocol A performs *better* (or *worse*) than B in the event that A returns fewer (alternatively *more*) rewards to the attacker than does B. Figure 5 supports the following conclusions. First, $\mathcal{B}_k$ performs worse than Bitcoin for low γ, but better than Bitcoin for high γ. The break-even point seems to be at $\gamma = 50\,\%$. Second, Tailstorm consistently outperforms Bitcoin and $\mathcal{B}_k$, even with constant rewards (TS/const). Moreover, discounting of rewards consistently makes Tailstorm less susceptible to selfish mining and similar incentive layer attacks. Third, the learned policies consistently match or outperform the hard-coded reference policies. This finding is important because anything else would indicate deficiencies in the training. Forth, the learned policies are close to optimal for Bitcoin. This can be seen in two ways. Firstly, Sapirshtein et al. showed that the SM1 policy is close to optimal for $\gamma = 0$ [47, Fig. 1a]. Our figure for $\gamma = 5\%$ shows that the learned policy matches SM1 (the dashed blue curve with $\times$-markers). Secondly, the authors showed that the optimal policy reaches the upper bound $\alpha/(1-\alpha)$ for $\gamma = 1$ [47, Fig. 1c]. Our figure for $\gamma = 95\%$ shows that the learned policy matches this bound as well.

Until now, we have evaluated the efficacy of policies in absolute terms. We now take a different perspective and ask: how strong must an attacker be for dishonest behaviour to pay off? Recall that honest behaviour implies an expected normalized reward of α (solid gray line in Figures 4 and 5). To answer the question, we calculate break-even points, which represent the minimum relative hash rate α, such that following the policy produces more than α of normalized reward. We start from the optimal policies presented in Figure 5 and select only those policies that feature dishonest behavior. Recall that each of the remaining policies was trained on a fixed relative hash rate α. For each protocol and choice of γ, we select the dishonest policy trained for the lowest α. We then evaluate this policy against alternative α values ranging from $5\,\%$ to $50\,\%$. Our simulator provides noisy observations and the reward distribution varies stochastically. Hence, we use Bayesian optimization to minimize the difference between the observed reward and α, i. e., to get as close to the

Table 2 Break-even points: minimal α (in %) where dishonest behaviour (Figure 5) pays off.

	Trained Policy			Reference Policy		
γ	5 %	50 %	95 %	5 %	50 %	95 %
Bitcoin	33.6	26.5	≤ 5	32.7	24.5	≤ 5
TS/const	28.9	28.6	21.7	37.9	32.6	28.4
Tailstorm	41.6	35.8	35.8	43.7	41.0	39.3
$\mathcal{B}_k$	22.2	27.8	25.3	31.3	31.1	30.7

break-even point as possible. Table 2 reports the results. Observe that for the trained policies, the break-even points are consistently either lower or close to the break-even points of the hard-coded reference policies. $\mathcal{B}_k$, TS/const, and Tailstorm are also less sensitive to changes in γ than Bitcoin. Tailstorm has the highest break-even points among all protocols, indicating that it is most resilient to incentive layer attacks.

7 Tailstorm Cryptocurrency

So far, we have focused on Tailstorm's consensus and incentive mechanisms. Throughout our analysis, we have assumed a cryptocurrency on the application layer that facilitates the creation and distribution of rewards to participants of the consensus layer. In this section, we describe and discuss the Tailstorm cryptocurrency and our prototype implementation.

7.1 Transaction Handling

Tailstorm implements the unspent transaction output (UTXO) model as it is used in Bitcoin [39]: Each UTXO represents a designated amount of cryptocurrency. Transactions consume and create UTXOs, but they never create more cryptocurrency than they consume. Consumed UTXOs cannot be consumed again. Ownership and transfer of value follows from restricting the consummation of UTXOs to the holders of specific cryptographic keys.

Tailstorm uses the same public key cryptography as Bitcoin, and it also supports Bitcoin's UTXO scripting facility. The dissemination and processing of transactions follows Bitcoin's approach as well. However, as we describe next, Tailstorm deviates from Bitcoin in how transactions are recorded in the blockchain.

Taking inspiration from Storm [4], each subblock contains a list of transactions. Let $\texttt{tx}_\text{a}$ be a transaction listed at position $\texttt{idx}_\text{a}$ in subblock $\texttt{sub}_\text{a}$ summarized in summary block $\texttt{sum}_\text{a}$ and let $\texttt{tx}_\text{b}$, $\texttt{idx}_\text{b}$, $\texttt{sub}_\text{b}$, and $\texttt{sum}_\text{b}$ be defined similarly. Assume $\texttt{tx}_\text{a}$ and $\texttt{tx}_\text{b}$ are incompatible, e.g., because they spend the same UTXO twice. If $\texttt{tx}_\text{a}$ and $\texttt{tx}_\text{b}$ are listed in the same subblock, i.e., $\texttt{sub}_\text{a} = \texttt{sub}_\text{b}$ and $\texttt{idx}_\text{a} \neq \texttt{idx}_\text{b}$, then this subblock is marked invalid. We proceed similarly if $\texttt{tx}_\text{a}$ and $\texttt{tx}_\text{b}$ are summarized by different summaries: due to Tailstorm's chain structure (see Alg. 2) $\texttt{sum}_\text{a}$ and $\texttt{sum}_\text{b}$ must differ in height and we mark the higher one as invalid. Honest nodes ignore invalid blocks and all their descendants at the consensus layer. Hence, under the above circumstances, incompatible transactions are not persisted in the blockchain, as is the case in Bitcoin.

However, special attention is required for incompatible transactions in different subblocks summarized by the same summary block, i.e., $\texttt{sub}_\text{a} \neq \texttt{sub}_\text{b}$ and $\texttt{sum}_\text{a} = \texttt{sum}_\text{b}$. A critical assumption of our analyses as well as parallel PoW in general [29] is that subblocks confirming the same summary are compatible and thus can be mined independently. We thus cannot mark the summary invalid in this case and the incompatible transactions are persisted in the blockchain. To resolve this conflict, Tailstorm executes only one of the transactions at the application layer and ignores the other according the following rules. Let $\texttt{dst}_\text{a}$ denote the distance of $\texttt{sub}_\text{a}$ to $\texttt{sum}_\text{a}$ in the block DAG, and let $\texttt{dst}_\text{b}$ be defined similarly. If $\texttt{dst}_\text{a} < \texttt{dst}_\text{b}$, then $\texttt{tx}_\text{a}$ is ignored. Ties are broken by the PoW hash function, i. e., if $\texttt{dst}_\text{a} = \texttt{dst}_\text{b}$ and $\texttt{hash}(\texttt{sub}_\text{a}) > \texttt{hash}(\texttt{sub}_\text{b})$, then $\texttt{tx}_\text{a}$ is ignored. If $\texttt{tx}_\text{a}$ is ignored then all dependent transactions (i.e. those spending the UTXOs produced by $\texttt{tx}_\text{a}$) are ignored as well. Note that even though incompatible transactions are persisted in the blockchain, the double-spending semantics are equivalent to Bitcoin: offending transactions off the longest branch are ignored.

7.2 Fast Confirmations

To reap the full consistency guarantees of parallel PoW [29], prudent cryptocurrency users should wait for one summary block confirmation before accepting their transactions as final. For example, if their transaction is included in subblock $\texttt{sub}_\text{a}$ and first summarized in $\texttt{sum}_\text{a}$, then they should wait for another valid summary $\texttt{sum}_\text{b}$ with $\texttt{height}(\texttt{sum}_\text{b}) = \texttt{height}(\texttt{sum}_\text{a})+1$. Assuming a 10 minute summary block interval and large k, the full confirmation will likely occur in 10 to 20 minutes, depending on whether the transaction was included early in the subblock tree or later.

If time is short and the transacted value is low, e. g. if the user is selling a cup of coffee to go, they can consider waiting for a number of subblock confirmations instead. For example, consider Tailstorm with $k = 60$ and a 10 minute summary block interval. Subblocks will be mined every 10 seconds in expectation. The heuristics are similar to a fast version of Bitcoin: If the seller waits for 6 subblock confirmations, i. e., for a subblock $\texttt{sub}_\text{b}$ with $\texttt{depth}(\texttt{sub}_\text{b}) = \texttt{depth}(\texttt{sub}_\text{a}) + 6$, the settlement will take about one minute. Invalidation of the payment, e. g. due to a double spend, implies a fork of the subblock tree of length 6 or more. Tailstorm discounts mining rewards according to the depth of the subblock tree; whereas the tree could have achieved depth 60, it now can achieve depth at most 54. One tenth of the minted rewards is lost, and the cost of the coffee is dwarfed.

7.3 Tailstorm Prototype

We have implemented Tailstorm and make the code available online [23]. We started from a fork Bitcoin Unlimited's implementation of Bitcoin Cash. Our fork implements the Tailstorm consensus layer and incentive mechanism as specified in Section 3. The prototype uses $k = 3$, but is easily configurable by changing a compile-time flag. The DAA is calibrated for a target summary block interval of ten minutes. Subblocks are expected to arrive every 200 seconds.

To minimize propagation delays, we have implemented several network compression mechanisms. First, we adapt the Graphene protocol [41] to avoid redundant transmission of transactions. Second, we adapt the Compact Block protocol [12] to reduce the size of summary blocks. Third, the transaction list is encoded bit-efficiently in each subblock.

We leave the implementation of Tailstorm's application layer, i. e. the transaction handling as described in Section 7.1, for future work. Finally, we note that our implementation has minimal testing and, being a prototype, is not ready for production use.

8 Discussion

Tailstorm inherits the consistency guarantees of parallel PoW consensus [29]. To increase fairness, we discount rewards based on the tree structure of subblocks which itself is inspired from Bobtail [9]. To obtain fast confirmations, we write transactions into subblocks like

Storm [4]. Alongside these foundational influences, we incorporate insights from a variety of related works, which we here can only list partially. For more details we refer to the surveys of Garay and Kiayias [19], describing the different ways of defining the consensus problem, and Bano et al. [6], focusing on the different solution approaches and protocols.

Tailstorm improves fairness by avoiding orphans and discounting rewards. The former is not new [2, 50, 51], however, the motivation differs. Sompolinsky and Zohar [51] propose to improve Bitcoin's *consistency* by referencing blocks that would otherwise be orphaned (called *uncles*) and then counting both blocks and uncles (GHOST-rule) whereas Bitcoin counts only the blocks (longest chain rule). However, the authors explicitly avoid rewarding uncles and hence do not improve *fairness*. Follow-up work [2, 50] does not discuss rewards at all. In Ethereum PoW, a deployed variant of GHOST [51], and Kaspa, a deployed variant of GHOSTDAG [50], uncles receive *partial* rewards and the blocks that include uncles get *additional* rewards. The unfair dynamic of Bitcoin, where included blocks receive the full reward and orphans none, is largely preserved. We think that GHOST-like protocols can become more fair by applying Tailstorm's ideas: discounting of rewards, applied equally to all involved miners.

Pass and Shi [42] explicitly set out to increase fairness in PoW. Their Fruitchains protocol uses two kinds of blocks like Tailstorm. Blocks record fruits and fruits record transactions. Unlike in Tailstorm, both fruits and blocks require PoW. Applying the *2-for-1 trick* of Garay et al. [20], both are mined with the same hash puzzle observing different bits of the output. Fruits have a lower difficulty and thus are created more frequently. The rewards are distributed evenly across recent fruits while blocks receive nothing. According to Zhang and Preneel [54], Fruitchains are more vulnerable to incentive layer attacks than Bitcoin for block race advantage $\gamma = 0$, and thereby less fair. Furthermore, Fruitchains suffer from a tradeoff between fairness and transaction confirmation time [54]. A related line of research [5, 53] extends the *2-for-1* into an *n-for-1* trick and makes the miners work on multiple chains in parallel. The chains are then interleaved to form a single coherent transaction ledger. Unlike Fruitchains, however, Prism [5] and OHIE [53] focus on consistency, liveness, and throughput. We leave for future work to investigate whether and how the fairness of these protocols can be improved by applying Tailstorm's reward discounting.

Kiffer and Rajaraman [33] propose to discount rewards inversely proportional to the overall hash rate participating in the system. This reduces centralization in their model, but we worry that motivating lower hash rates might harm the primary goals of consensus: consistency and liveness. Their mechanism poses an alternative to Bitcoin's halving mechanism, which statically reduces block rewards by 50 % roughly once every 4 years, to avoid long term inflation of the cryptocurrency. In contrast, Tailstorm's reward discounting is focused on the short term and punishes non-linearities within a single subblock tree.

But our approach is not without limitations. In Section 4, we analyze the impact of message propagation delays on fairness. We assume that these delays are uniform, affecting all miners equally. However, in practice, propagation delays depend on the connectivity of individual miners. Some inequalities arise from economies of scale, larger miners can invest more in low latency connections, others from the underlying physics of information propagation: miners located in the same region or on the same continent implicitly form are cartel, whereas joining the network from a distant location puts them at a disadvantage. We leave for future work to analyze how Tailstorm is affected by more realistic network topologies [1, 14, 38, 44, 45].

In Section 6, we perform a search for optimal attack strategies. Our search algorithm is based on reinforcement learning (RL), which means it is not exhaustive. It is possible that the RL agent did not discover the optimal strategy against certain protocols. This challenges our conclusion that Tailstorm is less susceptible to incentive layer attacks than the other protocols, Bitcoin and $\mathcal{B}_k$. To encourage further exploration we release the RL Gym environment on PyPI [28], allowing others to discover more effective attacks. Assuming the absence of better strategies, our results can be confirmed by encoding our attack space as a Markov Decision Process (MDP) and employing exhaustive search techniques, as has been done for Bitcoin [21, 47], Ethereum [55] and other longest chain protocols [54]. However, to the best of our knowledge, such techniques have not yet been successfully applied to DAG-based protocols like Tailstorm.

Another limitation is that our action space might not cover all possible attacks. We closely follow the modeling of the selfish mining attack space against Bitcoin [47], where we observe consent that the four actions `Adopt`, `Match`, `Override`, and `Wait` are indeed complete. We are convinced that adding two actions for summary formation, `Inclusive` and `Exclusive`, is enough to represent the most profitable attacks. However, our conclusions can be challenged by presenting a more effective incentive layer attack against Tailstorm, which cannot be expressed in our attack space. We encourage this endeavour by making all analytical code available online [30]. After modifying the attack space, new strategies can be drafted, evaluated, and optimized, while the virtual environment, protocol specifications, and RL attack search can be reused without changes.

On a separate note, Carlsten et al. [11] demonstrate that selfish mining becomes more profitable when considering transaction fees in addition to mining rewards. They present a strategy targeting Bitcoin which leverages transaction fees to outperform honest behavior for any $\alpha > 0$ and $\gamma < 1$. Similar attacks are likely feasible against all PoW cryptocurrencies, including Tailstorm, however they also exceed the scope of this paper.

Lastly, Arnosti and Weinberg [3] show that even small cost imbalances or economies of scale lead to highly centralized PoW mining ecosystems. While, Tailstorm reduces the imbalances compared to other PoW protocols we studied, it cannot fully mitigate centralization. Our notion of fairness revolves around rewarding miners in proportion to their hash rate. This definition is not new [1, 7, 31, 42, 54]. However, from an economic standpoint, to prevent centralization, miners should be rewarded in proportion to their operational mining costs [3]. Unfortunately, achieving this goal seems to be impossible. Mining costs primarily depend on the price and the availability of energy and specialized mining hardware. Purchasing energy in large quantities tends to be more cost-effective, and strong miners may even consider operating their own power plants. Scaling up operations also enables miners to develop and deploy more efficient mining hardware, and then selling it to weak miners after it has become outdated [52]. Given that these unfair scaling effects do not only affect Tailstorm, but PoW cryptocurrencies in general, one might be lead to consider alternative approaches like proof-of-stake [22, 32, 49]. But even then, it remains questionable whether any permissionless system can fully avoid centralization [36].

9 Conclusion

Tailstorm integrates parallel PoW [29], partial transaction confirmation [4, 9], and a novel incentive mechanism – reward discounting – into a PoW cryptocurrency that provides fast and secure confirmations for its users and fair rewards for its miners. We thereby solve a long standing issue of longest chain protocols, where the operator has to choose between either a short block interval with fast but less reliable confirmations and unfair rewards, or a long block interval with slow confirmations and more fair but infrequent rewards. Our prototype demonstrates that Tailstorm can replace Bitcoin not only theoretically but also in practice.

Contribution Statement

George originally developed the idea of reward discounting as a way to combat withholding attacks in Bobtail. George and Gregory refined that idea into an early version of Tailstorm suitable for a hard fork of Bitcoin Cash. They also contributed to the prototype implementation. Patrik contributed the protocol specification, simulation-based analyses, and reinforcement learning interface. George contributed the analytical orphan rate analysis. George and Patrik contributed the hard-coded reference policies. Ben and Patrik conducted the attack search with reinforcement learning. Guided by earlier stages of the analyses, Patrik provided improvements to Tailstorm consensus and its transaction handling. Patrik and George wrote this paper.

References

1 Mohamed Alzayat, Johnnatan Messias, Balakrishnan Chandrasekaran, Krishna P. Gummadi, and Patrick Loiseau. Modeling coordinated vs. P2P mining: An analysis of inefficiency and inequality in proof-of-work blockchains. *CoRR*, abs/2106.02970, 2021. `arXiv:2106.02970`.

2 Ignacio Amores-Sesar, Christian Cachin, and Anna Parker. Generalizing weighted trees: a bridge from bitcoin to GHOST. In Foteini Baldimtsi and Tim Roughgarden, editors, *AFT '21: 3rd ACM Conference on Advances in Financial Technologies, Arlington, Virginia, USA, September 26 - 28, 2021*, pages 156–169. ACM, 2021. `doi:10.1145/3479722.3480995`.

3 Nick Arnosti and S. Matthew Weinberg. Bitcoin: A natural oligopoly. *Manag. Sci.*, 68(7):4755–4771, 2022. `doi:10.1287/mnsc.2021.4095`.

4 awemany. Storm. `https://github.com/awemany/storm-sim/blob/master/whitepaper/`, 2019.

5 Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 585–602. ACM, 2019. `doi:10.1145/3319535.3363213`.

6 Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Sok: Consensus in the age of blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 183–198. ACM, 2019. `doi:10.1145/3318041.3355458`.

7 Georgios Birmpas, Elias Koutsoupias, Philip Lazos, and Francisco J. Marmolejo Cossío. Fairness and efficiency in dag-based cryptocurrencies. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2020. `doi:10.1007/978-3-030-51280-4_6`.

8 George Bissias. Radium: Improving dynamic pow targeting. In Joaquín García-Alfaro, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomartí, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2020 International Workshops, DPM 2020 and CBT 2020, Guildford, UK, September 17-18, 2020, Revised Selected Papers*, volume 12484 of *Lecture Notes in Computer Science*, pages 374–389. Springer, 2020. `doi:10.1007/978-3-030-66172-4_24`.

9 George Bissias and Brian Neil Levine. Bobtail: Improved blockchain security with low-variance mining. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020.* The Internet Society, 2020. URL: `https://www.ndss-symposium.org/ndss-paper/bobtail-improved-blockchain-security-with-low-variance-mining/`.

10 Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. `arXiv:1606.01540`.

11 Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 154–167. ACM, 2016. `doi:10.1145/2976749.2978408`.

12 Matt Corallo. BIP152: Compact Block Relay. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki, 2016.

13 Francisco J. Marmolejo Cossío, Eric Brigham, Benjamin Sela, and Jonathan Katz. Competing (semi-)selfish miners in bitcoin. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 89–109. ACM, 2019. `doi:10.1145/3318041.3355471`.

14 Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. Txprobe: Discovering bitcoin's network topology using orphan transactions. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, pages 550–566. Springer, 2019. `doi:10.1007/978-3-030-32101-7_32`.

15 Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and nakamoto always wins. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 859–878. ACM, 2020. `doi:10.1145/3372297.3417290`.

16 Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*, pages 436–454. Springer, 2014. `doi:10.1007/978-3-662-45472-5_28`.

17 Chen Feng and Jianyu Niu. Selfish mining in ethereum. In *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*, pages 1306–1316. IEEE, 2019. `doi:10.1109/ICDCS.2019.00131`.

18 Daniel Fullmer and A. Stephen Morse. Analysis of difficulty control in bitcoin and proof-of-work blockchains. In *57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018*, pages 5988–5992. IEEE, 2018. `doi:10.1109/CDC.2018.8619082`.

19 Juan A. Garay and Aggelos Kiayias. Sok: A consensus taxonomy in the blockchain era. In Stanislaw Jarecki, editor, *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, volume 12006 of *Lecture Notes in Computer Science*, pages 284–318. Springer, 2020. `doi:10.1007/978-3-030-40186-3_13`.

20 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015. `doi:10.1007/978-3-662-46803-6_10`.

21 Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 3–16. ACM, 2016. `doi:10.1145/2976749.2978341`.

22 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68. ACM, 2017. `doi:10.1145/3132747.3132757`.

23 Greg Griffith and George Bissias. Tailstorm node implementation. `https://gitlab.com/georgebissias/BCHUnlimited/-/tree/tailstorm_prototype`, 2023.

24 Dongning Guo and Ling Ren. Bitcoin's latency-security analysis made simple. In Maurice Herlihy and Neha Narula, editors, *Proceedings of the 4th ACM Conference on Advances in Financial Technologies, AFT 2022, Cambridge, MA, USA, September 19-21, 2022*, pages 244–253. ACM, 2022. `doi:10.1145/3558535.3559791`.

25 Thomas M. Harding. Real-time block rate targeting. *Ledger*, 5, 2020. `doi:10.5195/ledger.2020.195`.

26 Charlie Hou, Mingxun Zhou, Yan Ji, Phil Daian, Florian Tramèr, Giulia Fanti, and Ari Juels. Squirrl: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021.* The Internet Society, 2021. URL: `https://www.ndss-symposium.org/ndss-paper/squirrl-automating-attack-analysis-on-blockchain-incentive-mechanisms-with-deep-reinforcement-learning/`.

27 Geir Hovland and Jan Kucera. Nonlinear feedback control and stability analysis of a proof-of-work blockchain. *Modeling, Identification and Control*, 38(4):157–168, 2017. `doi:10.4173/mic.2017.4.1`.

28 Patrik Keller. PyPI release of our reinforcement learning environment as OpenAI Gym. `https://pypi.org/project/cpr-gym/0.7.0/`, 2023.

29 Patrik Keller and Rainer Böhme. Parallel proof-of-work with concrete bounds. In Maurice Herlihy and Neha Narula, editors, *Proceedings of the 4th ACM Conference on Advances in Financial Technologies, AFT 2022, Cambridge, MA, USA, September 19-21, 2022*, pages 1–15. ACM, 2022. `doi:10.1145/3558535.3559773`.

30 Patrik Keller and Ben Glickenhaus. Source code for our simulator, evaluations, and attack search. `https://github.com/pkel/cpr/tree/aft23`, 2023.

31 Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain mining games. In Vincent Conitzer, Dirk Bergemann, and Yiling Chen, editors, *Proceedings of the 2016 ACM Conference on Economics and Computation, EC '16, Maastricht, The Netherlands, July 24-28, 2016*, pages 365–382. ACM, 2016. `doi:10.1145/2940716.2940773`.

32 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388. Springer, 2017. `doi:10.1007/978-3-319-63688-7_12`.

33 Lucianna Kiffer and Rajmohan Rajaraman. Happy-mine: Designing a mining reward function. In Nikita Borisov and Claudia Díaz, editors, *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II*, volume 12675 of *Lecture Notes in Computer Science*, pages 250–268. Springer, 2021. `doi:10.1007/978-3-662-64331-0_13`.

34 Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. A better method to analyze blockchain consistency. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 729–744. ACM, 2018. `doi:10.1145/3243734.3243814`.

35 Daniel Kraft. Difficulty control for blockchain-based consensus systems. *Peer-to-Peer Netw. Appl.*, 9(2):397–413, 2016. `doi:10.1007/s12083-015-0347-x`.

36 Yujin Kwon, Jian Liu, Minjeong Kim, Dawn Song, and Yongdae Kim. Impossibility of full decentralization in permissionless blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 110–123. ACM, 2019. `doi:10.1145/3318041.3355463`.

37 Jing Li, Dongning Guo, and Ling Ren. Close latency-security trade-off for the nakamoto consensus. In Foteini Baldimtsi and Tim Roughgarden, editors, *AFT '21: 3rd ACM Conference on Advances in Financial Technologies, Arlington, Virginia, USA, September 26 - 28, 2021*, pages 100–113. ACM, 2021. `doi:10.1145/3479722.3480992`.

38 Sami Ben Mariem, Pedro Casas, Matteo Romiti, Benoit Donnet, Rainer Stütz, and Bernhard Haslhofer. All that glitters is not bitcoin - unveiling the centralized nature of the BTC (IP) network. In *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*, pages 1–9. IEEE, 2020. `doi:10.1109/NOMS47738.2020.9110354`.

39 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008.

40 Kevin Alarcón Negy, Peter R. Rizun, and Emin Gün Sirer. Selfish mining re-examined. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 61–78. Springer, 2020. `doi:10.1007/978-3-030-51280-4_5`.

41 A. Pinar Ozisik, Gavin Andresen, Brian Neil Levine, Darren Tapp, George Bissias, and Sunny Katkuri. Graphene: efficient interactive set reconciliation applied to blockchain propagation. In Jianping Wu and Wendy Hall, editors, *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*, pages 303–317. ACM, 2019. `doi:10.1145/3341302.3342082`.

42 Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 315–324. ACM, 2017. `doi:10.1145/3087801.3087809`.

43 Peter R. Rizun. Subchains: A technique to scale bitcoin and improve the user experience. *Ledger*, 1:38–52, 2016. URL: `https://ledgerjournal.org/ojs/index.php/ledger/article/view/40`.

44 Elias Rohrer and Florian Tschorsch. Kadcast: A structured approach to broadcast in blockchain networks. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 199–213. ACM, 2019. `doi:10.1145/3318041.3355469`.

45 Elias Rohrer and Florian Tschorsch. Blockchain layer zero: Characterizing the bitcoin network through measurements, models, and simulations. In *46th IEEE Conference on Local Computer Networks, LCN 2021, Edmonton, AB, Canada, October 4-7, 2021*, pages 9–16. IEEE, 2021. `doi:10.1109/LCN52139.2021.9524930`.

46 Sheldon M. Ross. *Introduction to Probability Models.* Elsevier, 2014. `doi:10.1016/C2012-0-03564-8`.

47 Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, volume 9603 of *Lecture Notes in Computer Science*, pages 515–532. Springer, 2016. `doi:10.1007/978-3-662-54970-4_30`.

48 John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. `arXiv:1707.06347`.

49 Jakub Sliwinski and Roger Wattenhofer. Asynchronous proof-of-stake. In Colette Johnen, Elad Michael Schiller, and Stefan Schmid, editors, *Stabilization, Safety, and Security of Distributed Systems - 23rd International Symposium, SSS 2021, Virtual Event, November*

17-20, 2021, Proceedings, volume 13046 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2021. doi:10.1007/978-3-030-91081-5_13.

50 Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. PHANTOM GHOSTDAG: a scalable generalization of nakamoto consensus: September 2, 2021. In Foteini Baldimtsi and Tim Roughgarden, editors, *AFT '21: 3rd ACM Conference on Advances in Financial Technologies, Arlington, Virginia, USA, September 26 - 28, 2021*, pages 57–70. ACM, 2021. doi:10.1145/3479722.3480990.

51 Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, volume 8975 of *Lecture Notes in Computer Science*, pages 507–527. Springer, 2015. doi:10.1007/978-3-662-47854-7_32.

52 Aviv Yaish and Aviv Zohar. Pricing ASICs for cryptocurrency mining. In *Proceedings of the 5th ACM Conference on Advances in Financial Technologies, AFT 2022, Princeton, NJ, USA, October 21-25, 2023*. LIPIcs, 2023.

53 Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. OHIE: blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 90–105. IEEE, 2020. doi:10.1109/SP40000.2020.00008.

54 Ren Zhang and Bart Preneel. Lay down the common metrics: Evaluating proof-of-work consensus protocols' security. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 175–192. IEEE, 2019. doi:10.1109/SP.2019.00086.

55 Roi Bar Zur, Ittay Eyal, and Aviv Tamar. Efficient MDP analysis for selfish-mining in blockchains. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 113–131. ACM, 2020. doi:10.1145/3419614.3423264.

STROBE: Streaming Threshold Random Beacons

Donald Beaver ✉
Independent Scholar, Pittsburgh, PA, USA

Konstantinos Chalkias ✉
Mysten Labs, Palo Alto, CA, USA

Mahimna Kelkar ✉
Cornell University, New York City, NY, USA

Lefteris Kokoris-Kogias ✉
Mysten Labs, London, UK
IST Austria, Klosterneuburg, Austria

Kevin Lewi ✉
Meta Platforms, Inc., Menlo Park, CA, USA

Ladi de Naurois ✉
Washington DC, USA

Valeria Nikolaenko ✉
a16z crypto, Palo Alto, CA, USA

Arnab Roy ✉
Mysten Labs, Palo Alto, CA, USA

Alberto Sonnino ✉
Mysten Labs, London, UK
University College London, UK

Abstract

We revisit decentralized random beacons with a focus on practical distributed applications. Decentralized random beacons (Beaver and So, Eurocrypt'93) provide the functionality for n parties to generate an unpredictable sequence of bits in a way that cannot be biased, which is useful for any decentralized protocol requiring trusted randomness.

Existing beacon constructions are highly inefficient in practical settings where protocol parties need to rejoin after crashes or disconnections, and more significantly where smart contracts may rely on arbitrary index points in high-volume streams. For this, we introduce a new notion of **history-generating decentralized random beacons** (HGDRBs).

Roughly, the *history-generation* property of HGDRBs allows for previous beacon outputs to be efficiently generated knowing only the current value and the public key. At application layers, history-generation supports registering a sparser set of on-chain values if desired, so that apps like lotteries can utilize on-chain values without incurring high-frequency costs, enjoying all the benefits of DRBs implemented off-chain or with decoupled, special-purpose chains. Unlike rollups, HG is tailored specifically to recovering and verifying pseudorandom bit sequences and thus enjoys unique optimizations investigated in this work.

We introduce STROBE: an efficient HGDRB construction which generalizes the original squaring-based RSA approach of Beaver and So. STROBE enjoys several useful properties that make it suited for practical applications that use beacons:

1. **history-generating**: it can regenerate and verify high-throughput beacon streams, supporting sparse (thus cost-effective) ledger entries;
2. **concisely self-verifying**: NIZK-free, with state and validation employing a single ring element;
3. **eco-friendly**: stake-based rather than work based;
4. **unbounded**: refresh-free, addressing limitations of Beaver and So;
5. **delay-free**: results are immediately available.
6. **storage-efficient**: the last beacon suffices to derive all past outputs, thus $O(1)$ storage requirements for nodes serving the whole history.

2012 ACM Subject Classification Security and privacy → Public key encryption

Keywords and phrases decentralized randomness, beacons, consensus, blockchain, lottery

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.7

Related Version *Full Version*: https://eprint.iacr.org/2021/1643

Acknowledgements Work done when all the authors were at Novi Research, Meta.

1 Introduction

A random beacon is a shared source of agreed-upon random bits. First introduced in 1981 by Rabin [44] in the context of digitally-signed documents, beacons use unpredictability to put adversarial strategies in doubt. Trusted beacons are increasingly recognized as a critical resource for Decentralized Finance (DeFi), blockchains, Byzantine Fault Tolerance (BFT), leader elections, and a range of Decentralized Applications (dApps) including lotteries.

At fundamental protocol layers such as consensus, beacons drastically reduce the time and effort needed to withstand protocol-defeating attacks. For example, Feldman and Micali [30] implemented a common coin to achieve 2-round probabilistic consensus, breaking the deterministic lower bound of $f + 1$ rounds [26]. As another example, most proof-of-stake blockchains use randomness to select leaders proposing blocks and, for increased efficiency, also use randomness to select subcommittees to confirm the blocks. To make sure the malicious actors have negligibly small window to corrupt the leaders or the subcommittees, it should be impossible for them to predict or bias the randomness used in the process, which creates strong demand for good randomness beacon protocols. Moreover, for the light-clients of these blockchains to synchronize the state, they need to verify the selection of leaders and subcommittees, implying that they have to verify the random beacon outputs.

At higher levels of abstraction, new and important properties and optimizations emerge. Crash failures, delays, and asynchrony in the interactions between processes or the execution of smart contracts in a ledger environment make it essential to have stronger and coordinated record-keeping, to make it easier to retrieve and verify past results.

To complicate matters, many high-level applications like lotteries or gaming require high-frequency streams. Frequencies and latencies at the scale of seconds are too slow. Registering streams bit-by-bit in real-time on a ledger is simply infeasible given transaction throughput and latencies, let alone enormously cost-prohibitive fees. Registering intermittent results is a feasible and cheaper workaround - as long as the intermittent results provide sufficient content and validation to enable optimistic fault detection and recourse for any on-chain relying parties. To date, such applications either run at very slow speeds (lotteries) or require centralized trust (gaming platforms).

While a naive approach might register a long sequence accompanied by aggregate signatures and rollups to prove step-by-step correctness, so that relying parties can check content and validity, we take a novel and alternative approach, employing **history generation** for streamlined communication size and speed. Our protocols are direct and simple, providing a concise element to *regenerate* the entire back sequence (effectively compressing it) and to *validate* it (obviating NIZKs, let alone aggregation).

Decentralized random beacons. Beaver and So presented the first decentralized beacon (DRB) [4], achieved by way of a threshold homomorphic secret sharing of future bits. Their approach capitalized on the Blum-Blum-Shub pseudorandom generator [7], in which successive squares in an RSA ring are produced. By reversing the sequence and employing tricks for threshold Lagrange interpolation, [4] gave a *self-certifying sequence* obviating any need for ZKPs.

History-generating DRBs. A key contribution of this work is the notion and implementation of *history generation*, a technique that is essentially equivalent to compressing long sequences and providing a maximally concise validity check. In the context of random streams produced by DRBs, we are able to take strong advantage of the sequential nature of the stream to achieve optimizations exceeding those available to general-purpose calculations-with-ZKPs.

Unbounded sequences. In the threshold homomorphic VSS approach of [4], there is a bound on the number of bits to be produced without engaging in some kind of refreshing state. The results are limited by stakeholders rather than VDF-style delays: there is no explicit notion of using work as a computationally-guaranteed limit on the cadence of a beacon. We address both limitations using a generalized construction without incurring proof-of-work.

Contributions. STROBE is a decentralized random beacon providing the following properties:

- **History-generating**: It can regenerate and verify high-throughput beacon streams, supporting sparse (thus cost-effective) ledger entries;
- **Concisely self-verifying**: It is NIZK-free, with state and validation employing a single ring element;
- **Eco-friendly**: It is stake-based rather than work-based;
- **Unbounded**: It is refresh-free, addressing limitations of Beaver and So;
- **Delay-free**: Results are immediately available.
- **Storage-efficient**: $O(1)$ storage for nodes serving the whole beacon history.

Roadmap. §2 describes intuitions of our constructions and how applications can benefit from the novel, history-generating property. We cover related work in §3, with particular attention to feature tradeoffs in Table 1. Details of the construction appear in §4, with security model in §5 and proofs in §6. We describe some extension in §7 and application details in §8.

2 Construction Intuition

Our starting point is to view RSA decryption as a trapdoor one-way function in reverse, which can be efficiently verified. The beacon output of an epoch is essentially the RSA decryption of the previous epoch's output, and this carries on perpetually. The verification of an output is to just RSA encrypt it with the public key and see that if it equals to the previous epoch's output. In this sense, the beacon is *self-certifying*.

Step 1: Beacon setup. Viewed in a non-distributed setting, the setup of the beacon generates RSA modulus $N = pq$ along with a root-ing parameter s. Then the beacon proceeds perpetually as:

$$x \to x^{1/s} \to x^{1/s^2} \to \cdots \to x^{1/s^T} \to \cdots$$

To parties that just know about N as a public parameter, this provides some attractive properties: (1) The next value in the beacon is hard to predict given the earlier values. (2) It's easy to verify a beacon value against the last value. In fact, we can check the value against any historical value, except that it gets progressively harder with the gap. (3) An especially tantalizing property is that any historical beacon value, can in fact be simply *generated* from the knowledge of the current value.

Step 2: Removing the trusted party. Of course, the problem with this is that taking roots in an RSA ring is hard without knowing the prime factors p and q. So we need a trusted party holding the primes to be generating all the beacon values. Recent advances in distributed RSA modulus generation allows us to generate a public modulus N, with no single party knowing the factors. Imagine that in addition we also give secret shares of s^{-1} $(mod\ \phi(N))$ to n distinct parties:

$$sk_1 + sk_2 + \cdots + sk_n = s^{-1}\ (mod\ \phi(N))$$

At the time epoch $T+1$, the n parties can then output $x_T^{sk_i}$ each, where x_T is the output of the last epoch. On multiplying all the public shares, we get $x_{T+1} = x_T^{s^{-1}}$ $(mod\ N)$. At the same time, observe that even $(n-1)$ of the secret keys are effectively independently random. Also observe every public share is also self-certifying wrt the last epoch, in the sense $x_T^{sk_i} = (x_{T+1}^{sk_i})^s$.

Step 3: Adapting to the threshold setting. Adapting the n-of-n setting to a threshold t-of-n setting introduces additional challenges which do not affect known-order groups. Essentially, Shamir secret sharing involves fractional Lagrange interpolation coefficients which are efficient to compute to group elements if we know the order. However, this is not possible to do in the exponent of RSA group elements, as $\phi(N)$ is not public. We adapt and extend the techniques pioneered by [4] and also used by [48] to address this challenge. The core trick is to lift the Lagrange coefficients by a factor of $n!$, so that they are not fractional anymore - details in Section 4.

3 Related Work

Random Beacons. Practical bias-resistant random beacons producing regular series of random outputs typically follow one of three approaches: the first one uses publicly verifiable secret sharing (PVSS) mechanisms; the second uses verifiable random functions (VRFs), often in conjunction with threshold cryptography, and the third relies on verifiable delay functions (VDFs).

Beacons of the first type use the following blueprint design, they need 4 rounds for n participants to generate a random value. In the first round all nodes simultaneously secret-share a freshly generated random value s among the rest of the nodes. They do so by publishing the following values: n shares of s each encrypted under receiving party's public key, a commitment to the secret, and a non-interactive zero-knowledge proof that those were generated correctly. In the second round, the parties run some sort of consensus algorithm to agree which nodes did the sharing correctly and which failed. This can also be replaced with posting the shares on chain where anyone can verify the proofs independently, although this typically relies on consensus provided by a chain (rather than enabled by the beacon). In the third round, the parties reveal their secrets and in the fourth round for parties that withheld their secrets, shares of those secrets are broadcast for recovery. The resulting beacon's value is derived from the revealed or recovered secrets. The rounds can be pipelined to get regular random outputs at each round. Starting from the original proposal of Ouroboros [36] a sequence of papers (Scrape [17], Albatross [18], HydRand [47], RandHerd [49]), OptRand [6] has improved the communication, computation complexity per node and the verification complexity for beacon of this type. No centralized or trusted setup and standard cryptographic assumptions are the main advantages of those protocols. However, those protocols still remain communication-intensive, since they need to run the full protocol (including consensus) for every fresh random value and computation wise intensive, since every party needs to check that every secret-share has been correctly constructed. A public verifier needs $O(n)$ messages to verify each beacon's value, making it communication-wise expensive to verify a series of random values. A full chain of our beacon, in contrast, can be verified using only the current beacon's output and the public parameters.

Beacons of the second type rely on a setup phase where a secret key is generated in a distributed manner, after which homomorphic VSS and/or threshold signatures can be employed. The use of homomorphic VSS was pioneered in the first DRB in 1993 [4], where successive values of a BBS generator are revealed. The shares of square roots are, homomorphically and similar in spirit to threshold RSA, square roots of shares. The shares and the reconstructed values each act as verifiers of previous values (*viz.* by way of squaring). This particular solution suffers from a need to refresh to new sequence values periodically; it is not unbounded as-is.

In numerous other generalized approaches of this second style, nodes can use a unique threshold signature scheme, such as threshold BLS [10, 11] (tBLS) in Dfinity [34] and drand [28], or as threshold RSA (tRSA) signatures [48] in Cachin et al. [15]) to sign an agreed-upon progression of values (either block-hashes, or round-numbers, or a combination of those). More generally, a verifiable random function is built in these construction to which unique signatures are a particular instantiatiation, but other constructions are also possible, e.g. NSEC5 VRF [41] used by Chainlink. The main advantage of those protocols is efficiency (each party only sends a single message per beacon's value) and ease of public verifiability (current beacon's value can be verified using only public parameters). The disadvantages are a complicated setup phase (though straightforward when a trusted party is assumed), for example to generate the threshold key for tBLS over the internet it still takes elaborate protocols with O(n^4) communication complexity costs [35, 38] and $O(f)$ worse case run time. Alternatively, instead of a threshold key, the parties can use independently generated keys (as is currently being done in the Ethereum's RANDAO approach [12]), although unfortunately such beacons have some small degree of bias from the parties who may decide to withhold their messages, we therefore do not focus on such beacons here despite the fact that certain application (e.g. Ethereum's committees selection) may stay resilient to small bias of the beacon. We improve on the approach of tRSA by building a random beacon straight from the RSA assumption requiring no additional proofs for correctness of signature's shares. Our scheme naturally gives a novel property of history generation in contrast to existing approaches.

There is also a third approach that relies on a proof-of-delay mechanism [8]: either a block-hash is passed through a verifiable delay function (VDF), or the values submitted by the participants are passed through a VDF function to generate a random value. This approach makes it simpler to build unbiasable beacons, as the participants will not have enough time to see how the bias on their contributions would affect the resulting value of the beacon. Most prominent systems are RANDAO w. VDF [27], continuous VDF [29] and RandRunner [46]. But those approaches are highly computationally intensive for the prover and require precise estimates of concrete complexity, which are hard to predict in practice (competitions with high-reward incentives were set-up by Chia Networks (`chia.net`) and Ethereum (`ethereum.org`) in partnership with Protocol Labs (`protocol.ai`) to get concrete estimates of VDFs' complexity). In the presence of a quantum adversary a quantum-resistant VDF could be used to produce a quantum-resistant random beacon, e.g. Veedo [50].

Recent surveys [22, 45] systematize knowledge of randomness beacons and discuss the complexity and practicality of the constructions in more details. In Table 1 we compare selected beacons to STROBE which is the only one that provides the property of history generation. Self-cerifying beacons allow to inexpensively verify a beacon value against the previous one; refresh-free beacons allow for generation of indefinite sequence of random values per single setup; BA-free beacons do not use Byzantine-Agreement protocols. Beacons based on the RSA assumption require a trusted setup or a distributed RSA modulus generation, our protocol additionally requires the generation of an inverse of a public exponent. The VDF-based beacons in RSA groups also do require a trusted setup.

Table 1 Comparison among several beacon protocols.

	History gen	Self certifying	Single round	Refresh-free	BA-free	Setup-free	
Albatross [18], HydRand [47], OptRand [6], RandHerd [49]	✗	✗	✗	✓	✓	✓	PVSS
RandRunner [46], RANDAO++ [27], cVDF [29], Veedo [50]	✗	✗	✓	✓	✓	✗/ ✓	VDF
Dfinity [34], drand [28]	✗	✗	✓	✓	✓	DKG	tVRF
C03 [13], BS93 [4]	✗	✓	✓	✗	✓	mod	RSA
STROBE (this work)	✓	✓	✓	✓	✓	mod+ inv	RSA

Distributed generation of an RSA modulus and an inverse of a public exponent. The setup of our construction requires the generation of an RSA modulus N that is a product of two primes. The most common way of generating such a modulus in a centralized setting (a.k.a. with a trusted setup), is to randomly sample κ-bits integers running Miller-Rabin probabilistic primality tests on them [40, 43] until two primes are obtained with overwhelming probability, multiplying them gives a 2κ-bits bi-prime N. Since currently there is no known way to sample a bi-prime (using only public randomness) for which nobody knows the factors, the only way to alleviate the trusted setup is to distribute the generation of the bi-prime modulus via a dedicated multi-party computation protocol. Boneh and Franklin [9] initiated the study of distributed RSA modulus generation devising a protocol in a passive security model with honest majority. The follow-up protocol of Algesheimer, Camenisch and Shoup [1] devised a protocol for generation of N that is a product of two *safe* primes, passively secure with honest majority. The follow-up work has hardened the original Boneh-Franklin's protocol to be secure in the presence of actively malicious parties and honest majority [31]. The works mentioned above also generate an inverse (RSA decryption key) in a distributed manner. An improvement to this part of the protocol was also made by Catalano et al. [19]. A promising approach of getting rid of the setup phase altogether, that was proposed in RandRunner [46] and in the work of Damgård and Koprowski [24], can potentially get applied to this work.

4 The STROBE Protocol

We now define the syntax of a History Generating Decentralized Random Beacon (HGDRB) and describe our STROBE construction.

▶ **Definition 1** (HGDRB). *A History Generating Decentralized Random Beacon (HGDRB) is a set of algorithms* $(Setup, Gen, Eval, VerifyShare,\ Combine,\ Verify, Back)$*:*

Setup: $(\lambda, n, t) \rightarrow (pk, sk_1, \cdots, sk_n)$. *The Setup algorithm takes the security parameter λ and threshold parameters n and t. The scheme allows t-of-n reconstruction, with secret shares given to n parties. The output is a public key pk and secret shares $sk_1, \cdots, sk_n$.*

Gen: $pk \rightarrow x_0$. *The (one-time) Gen algorithm samples an initial random value x_0.*

Eval: $(sk_i, x_T) \rightarrow x_{T+1,i}$. *Each party takes the last epoch's (T) output x_T, computes and outputs a share of the next epoch's output $x_{T+1,i}$.*

VerifyShare: $(pk, x_{T,i}, x_{T+1,i}) \rightarrow \{0,1\}$. *The VerifyShare algorithm checks the epoch $T+1$ shares against the corresponding epoch T shares.*

Combine: $(pk, x_{T,P_1}, x_{T,P_2}, \ldots, x_{T,P_t}) \rightarrow x_{T+1}$. *Given t shares from epoch T that pass* VerifyShare *the Combine algorithm outputs the next epoch's beacon value x_{T+1}.*

Verify: $(pk, x_T, x_{T+1}) \rightarrow \{0,1\}$. *The Verify algorithm checks the epoch $T+1$ beacon output against the epoch T beacon output.*

Back: $(pk, x_T, k) \rightarrow x_{T-k}$: *This outputs the beacon value at epoch $T-k$ given the epoch T beacon value x_T and $k < T$.*

Informally, correctness asserts that honestly computed beacon values will pass verify checks with respect to previous beacon outputs. The same should hold for share outputs as well. The *Back* function allows to compute any historical beacon value efficiently (going back a polynomial number of epochs).

The STROBE protocol

We now provide our HGDRB construction, called *STROBE*. It is based on threshold inversion in RSA groups and its security follows from the RSA assumption.

Setup(λ, n, t)**:** The Setup algorithm takes the security parameter λ and samples an RSA modulus $N = pq$ with $\phi(N) = 4p'q'$.
Sample primes p, q such that $p - 1 = 2p', q - 1 = 2q'$ with p', q' also being primes. Pick a prime s, s.t. $\min(p', q') > s > n$. Let $N = pq$ and observe that $s \nmid \phi(N)$. Sample $a_1, \cdots, a_{t-1} \leftarrow [1, N]$ and let $f(X) = v + a_1 X + \cdots + a_{t-1} X^{t-1}$, where $v = (n!s)^{-1} \pmod{p'q'}$.
Send secret shares $sk_i = f(i) \pmod{\phi(N)}$ to parties $i \in [1, n]$. Publish $pk = (N, s)$.

Gen(pk)**:** The Gen algorithm samples a seed value $\mathsf{seed} \leftarrow [1, N]$. This is a public random value that can be computed by MPC or by taking a block hash. It then outputs $x_0 = \mathsf{seed}^{4(n!)^2} \pmod{N}$.
The $4(n!)^2$ factor is an artifact of the security proof and will be explained in Section 6.

Eval(sk_i, x_T)**:** Each party takes the last epoch's (T) output x_T, computes and outputs a share of the next epoch's output $x_{T+1,i}$:

$$x_{T+1,i} = x_T^{sk_i} = x_T^{f(i)} \pmod{N}.$$

VerifyShare$(pk, x_{T,i}, x_{T+1,i})$**:** The epoch $T + 1$ shares can be self-verified against the corresponding epoch T shares, by checking that

$$x_{T+1,i}^s = x_{T,i} \pmod N.$$

Combine$(pk, x_{T,P_1}, x_{T,P_2}, \ldots, x_{T,P_t})$**:** Given t epoch T shares, first check that each of them pass VerifyShare. Let γ denote the set of these indices $\{P_1, \cdots, P_t\}$. The Combine algorithm then computes the combined epoch T beacon value x_T by computing the interpolation:

$$x_T = \prod_{i \in \gamma} x_{T,i}^{n! L_i(0)} \pmod N,$$

where the Lagrange basis polynomials L_i's are defined as:

$$L_i(X) = \prod_{j\in\gamma, j\neq i} \frac{X-j}{i-j}.$$

The polynomials satisfy the following property: for $\forall i, j \in \gamma : L_i(i) = 1$, and $L_i(j) = 0$ for $i \neq j$. The $(n!)$ factor is essential to clear the denominators of the interpolation coefficients, which makes sure there is no fractional exponent to compute.

Verify(pk, x_T, x_{T+1}): The epoch $T + 1$ beacon output can be self-verified against the corresponding epoch T beacon output, by checking that

$$x_{T+1}^s = x_T \pmod N.$$

Back(pk, x_T, k): This outputs the beacon value at epoch $T - k$ as:

$$x_{T-k} = x_T^{s^k} \pmod N.$$

We assume that each share carries the identity information (metadata) about which party generated it. Apart from the $Verify$ (and $VerifyShare$) algorithms checking against the last epoch outputs, the *Back* algorithm provides an alternate way to check against any previous epoch output, all the way to epoch 0 (and 1). In fact if the shares from epoch 1 are also made part of the trusted pk, then this provides a way of checking the beacon value without accessing the past beacon values at all. The trade-off is that checking against beacon values in the past grows computationally expensive with the number of epochs elapsed. As an extension of the core protocol, we can also leverage techniques from popular VDF constructions, as described in Section 7. We now give the proof of correctness of the protocol.

4.1 Proof of Correctness of STROBE

We have $x_0 = \mathsf{seed}^{4(n!)^2}$. Assume inductively, $x_{T-1} = \mathsf{seed}^{4(n!)^2 s^{-T+1}}$. Correctness of the beacon outputs follows as below:

$$x_T = \prod_{i\in\gamma} x_{T,i}^{n! L_i(0)} = x_{T-1}^{n! \sum_{i\in\gamma} f(i)L_i(0)} = \mathsf{seed}^{4(n!)^2 s^{-T+1} n! \sum_{i\in\gamma} f(i)L_i(0)}$$

$$= \mathsf{seed}^{4(n!)^2 s^{-T+1} n! f(0)} = \mathsf{seed}^{(n!)^2 s^{-T} 4(n!)vs}$$

As $v = (n!s)^{-1} \pmod{p'q'}$, we have $4(n!)vs = 4 \pmod{\phi(N)}$. Therefore, $\mathsf{seed}^{4(n!)vs} = \mathsf{seed}^4 \pmod N$.

Substituting, we get:

$$x_T = \mathsf{seed}^{4(n!)^2 s^{-T}}.$$

This carries the induction successfully forward, and also leads to successful verification: $x_T^s = x_{T-1}$. Similar steps apply to the individual shares as well.

5 Security Model

There are various flavors of security that we could require of a random beacon. To narrow down the syntax, we will focus on *stake-based, self-certifying, threshold* beacons. The baseline security we want is that an adversary should not be able to predict future beacon values based on seeing past values and corrupting less than a threshold number of participants.

Unpredictability vs. Pseudorandomness

We could require the next beacon value to be pseudorandom, instead of just unpredictable. We observe that we could essentially compile an unpredictable beacon into a pseudorandom one, either by applying a random oracle (similarly to what is described as "tick-tock" in [29]), or if we want to avoid the RO assumption, by extracting hardcore bit(s), as in [7]. There exist applications where unpredictability suffices [21], but in most of the cases, such as a decentralized lottery or leader election, unbiased randomness is essential [45]. We also note that, for a self-certifying and/or history generating beacon, there needs to be a part of the beacon output that cannot be pseudorandom, as otherwise it cannot be used for the certification assertion and/or historical value computation.

Active vs. Passive

A passive adversary just observes the transcript of an honest run of the protocol, including public shares and beacon outputs and then tries to predict the next beacon value. An active adversary, on the other hand, can actively modify the public share values and beacon outputs. In this paper we consider active adversaries. In our setting, the self-certification essentially ensures that there is only a unique value for each expected public share or beacon value that can pass onto the next phase. This makes any adversarial modifications immediately noticeable and subject to rejection.

Selective vs. Adaptive

Another dimension to specify the adversary is on whether we restrict it to corrupt parties that it declares upfront (selective), or allow the parties to be corrupted dynamically as it observes and interacts with the protocol (adaptive). Our core protocol only satisfies selective security in this respect. We leave it as an open problem to construct an adaptively secure protocol which retains the efficiency standard of our selective one. A generic (complexity-leveraging) approach can be used to get an adaptively secure scheme from a selectively secure one: the reduction simply needs to guess the set of corrupted parties and then run the selective reduction aborting if the selection of corrupted parties does not match the one requested by the adversary. This, unfortunately, leads to a loss in security that is exponential in the number of parties, n, thus limiting the number of parties to be at most logarithmic in the security parameter λ. There are other promising approaches in the works for threshold RSA cryptosystems. Canetti et al. [16] proposed a methodology for transforming a selectively-secure threshold scheme into an adaptively-secure one, where the protocol needs to be modified to carefully erase secrets and to use simple zero-knowledge proofs, the adversary is rewinded in the proof which also incurs a security loss although not as large as with the complexity-leveraging approach. A follow-up work of Almansa et al. [2] simplified this result for RSA, but at the cost of the secret's re-sharing after every round, with an emergent benefit of making the scheme proactively secure. In a proactively secure scheme the adversary can corrupt at most t players in a time period determined by the protocol. Since the set of corrupted parties changes, each party can become corrupt at some point (i.e. leak its secrets), but if the party recovers from a compromise then a subsequent secret's re-sharing will enable the party to be honest again.

Given the above discussion, we now formally define the security model that we consider for our core protocol: unpredictable, passive and selective.

▶ **Definition 2** (Selective-secure Unpredictability). *We say that an HGDRB is* Selective-secure Unpredictable *if the following adversary has negligible advantage:*

1. *The Challenger runs Setup(λ, n, t) and outputs pk to the Adversary.*
2. *The Adversary selects a time epoch $S < poly(\lambda)$ and a set of parties $\gamma = \{P_1, P_2, \cdots, P_{t-1}\}$ to corrupt.*
3. *The Challenger sends the transcript of the protocol till time epoch S to the Adversary, as well as the secret shares for parties in γ. This includes the outputs of Gen and those of Eval for all $i \in [1, n]$ and $T \leq S$.*
4. *The Adversary outputs a quantity x'.*
5. *The Adversary wins if $Verify(pk, x_S, x')$ passes.*

In the next section, we formally prove that our construction satisfies Selective-Secure Unpredictability. We also claim that the construction protects against active adversaries, as the self-verification of beacon and share values ensure that only the unique correct values would not be rejected.

6 Proof of Security

In this section, we show that the STROBE protocol satisfies Selective-secure Unpredictability under the RSA assumption.

▶ **Definition 3** (RSA Assumption). *The Challenger samples RSA number $N = pq$ and picks a quantity s co-prime to $\phi(N)$. Then it randomly samples $z \leftarrow [1, N]$ and sends (N, s, z) to the Adversary. The Adversary outputs y. The RSA assumption states that the probability of $y^s = z \pmod N$ is negligible.*

▶ **Theorem 4** (Security). *The STROBE protocol is Selective-secure Unpredictable under the RSA Assumption.*

Proof. Let (N, s, z) be an RSA challenge for a prime $s > n$.

Setup. The Challenger outputs $pk = (N, s)$. Suppose the Adversary picks an epoch S and corrupts parties in the set $\gamma = \{P_1, \cdots, P_{t-1}\}$. Sample $b_{P_1}, \cdots, b_{P_{t-1}} \leftarrow [1, N]$. Send Adversary shares $sk_i = b_i$, for all $i \in \gamma$.

Eval. Consider an implicit polynomial $f(X)$, such that $f(0) = v$, where $v = (n!s)^{-1} \pmod{p'q'}$, and $f(i) = b_i$, for all $i \in \gamma$:

$$f(X) = vL_0(X) + \sum_{i \in \gamma} b_i L_i(X).$$

Here $L_i(X)$ are Lagrange basis polynomials of degree $t-1$ each:

$$L_i(X) = \prod_{j \in \{\gamma \cup \{0\}\}, j \neq i} \frac{X - j}{i - j}.$$

The polynomials satisfy the following property: for $\forall i, j \in (\gamma \cup \{0\}) : L_i(i) = 1$, and $L_i(j) = 0$ for $i \neq j$. Note that it is possible to evaluate $z^{n! L_i(j)}$ for any $i, j \in [0, n]$, since the $(n!)$ factor eliminates the denominators of interpolation polynomials, making it possible to evaluate the exponentiation.

Set $x_0 = z^{4(n!)^2 s^S}$ and $x_{T,j} = z^{4f(j)(n!)^2 s^{S-T+1}}$. Now, for $j \in \gamma$, we can compute $x_{T,j}$ explicitly based on $f(j) = b_j$. We now show that for $j \notin \gamma$, we can compute $x_{T,j}$ without explicitly computing $f(j)$. Observe that $4v = 4(n!s)^{-1} \pmod{\phi(N)}$), therefore $z^{v4n!s} = z^4$ hence $x_{T,j}$ can be explicitly constructed as follows:

$$x_{T,j} = z^{4f(j)(n!)^2 s^{S-T+1}} = z^{(f(j)4n!s)\cdot(n!)s^{S-T}} = \quad z^{(4L_0(j)+\sum_{i\in\gamma} b_i\cdot(4n!s)\cdot L_i(j))\cdot(n!)s^{S-T}}$$

Note that the last $(n!)$ factor is essential to make sure we can clear the denominator of $L_0(j)$ and thus avoid any divisions in the exponent. Send x_0 and the $x_{T,j}$'s for all $T \in [1, S]$ and $j \in [1, n]$ to the adversary.

RSA response. Let's suppose the adversary comes up with x' as the next epoch candidate. Given the self-certification checks upto time epoch T, we inductively have $x_k = x_0^{s^{-k}} = z^{4(n!)^2 s^{S-k}}$, for $k \in [0, S]$. If the adversary response x' passes verification, then we should have $x'^s = x_S = z^{4(n!)^2}$, Let $w = (4(n!)^2)^{-1} \pmod{s}$ and let $w(4(n!)^2) = 1 + ks$ for some computable integer k. Then $x'^{sw} = z^{4(n!)^2 w} = z^{1+ks}$. Therefore, $z = (x'^w z^{-k})^s$. Hence $x'^w z^{-k}$ will be a winning response to the RSA challenge.

Distributions. Finally, we observe that as s^S is invertible modulo $\phi(N)$ and sampling uniformly from $[1, N]$ and $[1, \phi(N)]$ are statistically indistinguishable, the distribution of x_0 in the STROBE construction and this proof are statistically indistinguishable. Matching these distributions is the technical reason behind the extra $4(n!)^2$ factor in the *Gen* algorithm. ◀

On active adversaries

An active adversary is allowed to tamper with the intermediate shares and beacon values. Observe that, as s is invertible with respect to $\phi(N)$, the beacon values are deterministic in the past as well as the future, given a current value. Therefore, given the initial value x_0, an adversarially tampered beacon value would be verifiably detected.

It is possible that intermediate shares could be tampered with without detection in certain circumstances. For example, if one of the parties is absent in all the epochs up to a certain point and then starts participating. The upshot of such attacks is essentially denial of service. The correct verifiability of beacon outputs is not affected.

7 Extensions

In this section we discuss some extensions of our core scheme to enable further functionalities and guarantees.

7.1 Dynamic Beacon Committees

STROBE can also easily handle dynamically changing the participants executing the beacon protocol. For example, this could involve rotating to a newly chosen committee (even of a different size) of parties after some fixed number of epochs. The new committee will then continue to generate future outputs of the beacon. In fact, the new committee can also be chosen based on the output of the distributed beacon. Dynamic participation can be accommodated by using the old committee to reshare the secret key to the new committee through existing literature on dynamic proactive secret sharing [5, 39]. Note however, that such protocols need to make the assumption that honest parties from previous committees delete their shares so that an adversary cannot recover the secret key even by corrupting more than a threshold number of parties from a previous committee. The above scenario is also referred to as a "long range" attack (c.f., [21]).

7.2 Succinct Proofs of Beacon Validation

In earlier sections we observed that it is possible to check the beacon value x_T at epoch T against the seed value x_0 by checking $x_0 = x_T^{s^T}$. However, this takes sequential time T. One way to speed up verification is to exploit the RSA repeated powering structure of this check and use existing techniques like [42, 51] to add a proof in addition to the beacon value.

In particular, we can just apply Wesolowski's [51] proof technique in reverse and produce the proof $\pi = x_T^{\lfloor s^T/\ell \rfloor}$, where ℓ is the result (a prime number) of a random oracle H applied to (x_0, x_T, T). The verifier computes ℓ and r as the remainder on dividing s^T by ℓ and then checks $x_0 = \pi^\ell x_T^r$. However, a drawback of this method is that producing the proof takes about T time as well, which may not be ideal to do every epoch.

A better approach is to use a continuous VDF [29]. In a continuous VDF, it is efficient to publish and use intermediate proofs at every time epoch. The Ephraim et al [29] continuous VDF makes use of the recursive structure of Pietrzak's VDF [42]. The high level idea is that they checkpoint a logarithmic number of past values and keep recursively merging an appropriate number of them in order to prevent growing the proof size. We can also similarly checkpoint and merge based on a similar recursion, while reversing the order: $x_0 = x_u^{s^{T/2}}$ and $x_u = x_T^{s^{T/2}}$, where $u = T/2$ is the midpoint.

We can use a continuous VDF at defined intervals as well, instead of every epoch. A verifier can sequentially compute the expected value until the last such interval and then just check the VDF proof.

7.3 Deeper Blockchain Integration

The design described in Section 4 relies on external authorities. However, incorporating STROBE within the infrastructure of a blockchain would enable the generation of randomness as a side effect of the normal system operations, taking no additional dependency on external authorities. Allowing efficient reconfiguration of STROBE for permissionless environments with highly dynamic set of nodes remains an open problem. Nevertheless, integration of STROBE into permissioned (BFT-based) blockchain platforms is straightforward. For instance, in Hyperledger Fabric [14], contracts run on private sets of computation nodes and use the Fabric protocols for cross-contract calls. In this setting, STROBE authorities can coincide with the Fabric smart contract authorities. Upon a contract setup, they perform a setup and key distribution, and then start generating randomness when authorized by the contract. For generating randomness, the only secrets maintained are the private STROBE authorities keys; all other operations of the contract can be logged and publicly verified. The threshold trust assumption – namely that of integrity and availability – is preserved.

Incentives for Randomness Generation

One open question is on managing the incentives of releasing shares for the randomness beacon. On a first sight the fault-tolerance of STROBE makes the problem a public goods game, since only a threshold needs to participate. One way to break this has been proposed in prior work [3, 37] where the shares are published on chain and rewards are given only to "useful" shares (*i.e.* the first t that make it on-chain). This breaks the public goods game and makes it a race between the authorities who are eager to release their share and get rewarded.

8 Applications

In this section, we discuss applications of random beacons. In particular, we argue how the novel history generation feature of STROBE can enable attractive technical advancements in many scenarios.

8.1 Blockchain Light-clients

Blockchain's light client [20] is typically a resource-constrained node that performs limited verification of the blockchain by downloading block headers and verifying high-level state transitions following consensus decisions. In the context of proof-of-stake blockchains, a light-client would verify the validators' signatures over block headers (in BFT-style protocols) or leader elections (in longest-chain-style protocols), or both (in hybrid-style blockchains), as well as track the changes to the validator set. Some protocols (e.g. Ethereum 2.0) do subsampling of the committees artificially lowering the number of parties in order to help drive the protocol to agreement faster. Both leader election and subsampling is done through the random-beacon protocols. This guarantees that the process is fair, transparent and verifiable, as well as that no malicious party can increase the probability of them being elected, and that no attacker can predict the next leaders or members of the subcommittees in advance preventing the launch of a targeted attack. All of these properties are provided by verifiability, uniformity, unbiasability and unpredictability of a random beacon. However, a major challenge for light-clients is to follow and verify the beacon outputs in order to be able to follow and verify consensus decisions. A protocol that allows to generate historical values of the beacon, like ours, would allow the light-client to only verify the latest output and be able to generate the prior values locally simplifying the process of verifying the leaders and subcommittees.

8.2 Blockchain-based Gambling and Lotteries

Lotteries and gambling smart contract solutions are gaining in popularity, and a portion of them advertise transparency, unbiased randomness generation (uncontrollable from participants) and consumer privacy. It is well known that the original success of Bitcoin was partly due to gambling activities, especially via the Satoshi Dice which operated since 2012 and dominated the bitcoin transactions in its first years of operation [32]. There is a growing number of blockchain gambling contracts, and as of September 2021, according to statistics provided in [25], there exist at least 151 lottery, 129 casino, 70 poker and generally 600+ gambling smart contracts in Ethereum alone. Moreover, lotteries have also been proposed as an alternative reward scheme for miners by randomly recirculating lost coins and collecting gold dust [33].

One of the main advantages of STROBE in this setting is that the latest beacon can be automatically used to "verifiably" derive all of the previous random numbers down to the original genesis beacon. This property can be utilized by smart contract developers to minimize cost by skipping beacon epochs when required, but lotteries can still continuously run for every epoch. It also works as a DoS defense, especially when the beacon is provided by an external oracle service to the blockchain; if there is significant delay on updates, the latest beacon suffices to execute all of the pending lottery games.

8.3 High-throughput Beacon Streams

There exist applications requiring constant high-throughput of beacons, especially in the online gaming sector. Most games require a combination of "skills" and "luck", and as video gaming has become a market where professional players participate in tournaments with real prizes, a fair beacon stream would enable new types of transparent gaming features. Due to potential internet speed issues, server overloading and other factors, reliability might be at stake [23] and thus, some game providers prefer UDP connections to offer greater flexibility by executing packets out of order or discarding non important ones [52]. STROBE's history generation feature fits really well in streaming designs, and allows client software to generate game states by computing every missing beacon. Note that in these applications, VDF-based beacons are not good candidates due to the intrinsic delay and high-latency, while the proof of stake nature of STROBE offers fairness guarantees by not relying in the event organizer's or software publisher's honesty. Along the same lines, STROBE has an advantage in low or expensive bandwidth locations (i.e., remote IoT devices) by allowing reading a beacon infrequently and generating past randomness internally.

Finally, STROBE allows for flexible and more efficient database requirements on nodes serving the API for the derived random beacon per round. Typically, only the last beacon suffices, thus $O(1)$ storage, while one can implement a service that maintains a wisely selected number of checkpoints for faster past-beacon lookups as a trade-off between memory/storage and computations.

References

1 Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *CRYPTO*, pages 417–432, 2002.

2 Jesús F Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold RSA with adaptive and proactive security. In *EUROCRYPT*, pages 593–611, 2006.

3 Zeta Avarikioti, EK Kogias, Roger Wattenhofer, and Dionysis Zindros. Brick: Asynchronous incentive-compatible payment channels. In *FC*, pages 209–230, 2021.

4 Donald Beaver and Nicol So. Global, unpredictable bit generation without broadcast. In *EUROCRYPT*, pages 424–434, 1993.

5 Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a blockchain keep a secret? In *TCC*, pages 260–290, 2020.

6 Adithya Bhat, Aniket Kate, Kartik Nayak, and Nibesh Shrestha. Optrand: Optimistically responsive distributed random beacons. In *NDSS*, 2023.

7 Lenore Blum, Manuel Blum, and Michael Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Comput.*, 15:364–383, 1986.

8 Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO*, pages 757–788, 2018.

9 Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In *CRYPTO*, pages 425–439, 1997.

10 Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.

11 Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT*, pages 514–532, 2001.

12 Vitalik Buterin. Vitalik's annotated ethereum 2.0 spec. `https://github.com/ethereum/annotated-spec/blob/master/phase0/beacon-chain.md#randao`, 2020.

13 Christian Cachin. An asynchronous protocol for distributed computation of RSA inverses and its applications. In *PODC*, pages 153–162, 2003.

14 Christian Cachin. Architecture of the Hyperledger blockchain Fabric. In *DCCL*, 2016.

15 Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

16 Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *CRYPTO*, pages 98–116, 1999.

17 Ignacio Cascudo and Bernardo David. SCRAPE: Scalable randomness attested by public entities. In *ACNS*, pages 537–556, 2017.

18 Ignacio Cascudo and Bernardo David. ALBATROSS: publicly attestable batched randomness based on secret sharing. In *ASIACRYPT*, pages 311–341, 2020.

19 Dario Catalano, Rosario Gennaro, and Shai Halevi. Computing inverses over a shared secret modulus. In *EUROCRYPT*, pages 190–206, 2000.

20 Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. SoK: Blockchain light clients. In *FC*, 2022.

21 Panagiotis Chatzigiannis and Konstantinos Chalkias. Proof of assets in the Diem blockchain. In *ACNS*, pages 27–41, 2021.

22 Kevin Choi, Aathira Manoj, and Joseph Bonneau. SoK: Distributed randomness beacons. In *IEEE S&P*, pages 75–92, 2023.

23 Mia Consalvo. *Cheating: Gaining advantage in videogames*. Mit Press, 2009.

24 Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In *EUROCRYPT*, pages 152–165, 2001.

25 Dapp.com. List of gambling Ethereum smart contracts, 2021. URL: `https://www.dapp.com/search_product?keyword=gambling`.

26 Danny Dolev and H Strong. Polynomial algorithms for multiple processor agreement. *SIAM J Computing*, 12(4):656–666, 1982.

27 J. Drake. Minimal VDF randomness beacon. URL: `https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566`.

28 Drand - a distributed randomness beacon daemon. URL: `https://drand.love/`.

29 Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In *EUROCRYPT*, pages 125–154, 2020.

30 Paul Feldman and Sylvio Micali. Byzantine agreement in constant expected time. In *FOCS*, pages 267–276, 1997.

31 Yair Frankel, Philip D MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *STOC*, pages 663–672, 1998.

32 Sally M Gainsbury and Alex Blaszczynski. How blockchain and cryptocurrency technology could revolutionize online gambling. *Gaming Law Review*, 21(7):482–492, 2017.

33 Harald Gjermundrød, Konstantinos Chalkias, and Ioanna Dionysiou. Going beyond the coinbase transaction fee: Alternative reward schemes for miners in blockchain systems. In *PCI*, pages 1–4, 2016.

34 Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

35 Aniket Kate and Ian Goldberg. Distributed key generation for the internet. In *ICDCS*, pages 119–128, 2009.

36 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, pages 357–388, 2017.

37 Eleftherios Kokoris-Kogias, Enis Ceyhun Alp, Linus Gasser, Philipp Jovanovic, Ewa Syta, and Bryan Ford. Calypso: Private data management for decentralized ledgers. *Proc. VLDB Endow.*, 14(4):586–599, 2020.

38 Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *CCS*, pages 1751–1767, 2020.

39 Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. Churp: Dynamic-committee proactive secret sharing. In *CCS*, pages 2369–2386, 2019.

40 Gary L Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

41 Dimitrios Papadopoulos, Duane Wessels, Shumon Huque, Moni Naor, Jan Včelák, Leonid Reyzin, and Sharon Goldberg. Making nsec5 practical for dnssec. *Cryptology ePrint Archive*, 2017.

42 Krzysztof Pietrzak. Simple verifiable delay functions. In *ITCS*, pages 60:1–60:15, 2019.

43 Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.

44 Michael O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, 1983.

45 Mayank Raikwar and Danilo Gligoroski. Sok: Decentralized randomness beacon protocols. *arXiv preprint arXiv:2205.13333*, 2022.

46 Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. Randrunner: Distributed randomness from trapdoor VDFs with strong uniqueness. In *NDSS 2022*, 2021.

47 Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. Hydrand: Efficient continuous distributed randomness. In *IEEE S&P*, pages 73–89, 2020.

48 Victor Shoup. Practical threshold signatures. In *EUROCRYPT 2000*, pages 207–220, 2000.

49 Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE S&P*, pages 444–460, 2017.

50 VeeDo is a STARK-based verifiable delay function (VDF) service. URL: `https://github.com/starkware-libs/veedo`.

51 Benjamin Wesolowski. Efficient verifiable delay functions. In *EUROCRYPT*, pages 379–407, 2019.

52 Zheng Xue, Di Wu, Jian He, Xiaojun Hei, and Yong Liu. Playing high-end video games in the cloud: A measurement study. *IEEE Trans. Cir. and Sys. for Video Technol.*, 25(12):2013–2025, 2014.

User Participation in Cryptocurrency Derivative Markets

Daisuke Kawai
Carnegie Mellon University, Pittsburgh, PA, USA

Bryan Routledge
Carnegie Mellon University, Pittsburgh, PA, USA

Kyle Soska
Ramiel Capital, New York, NY, USA

Ariel Zetlin-Jones
Carnegie Mellon University, Pittsburgh, PA, USA

Nicolas Christin
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

As cryptocurrencies have been appreciating against fiat currencies, global markets for cryptocurrency investment have started to emerge, including, most prominently, derivative exchanges. Different from traditional derivative markets, cryptocurrency derivative products are directly marketed to consumers, rather than through brokerage firms or institutional investors. Cryptocurrency derivative exchange platforms include many game-like features (e.g., leaderboards, chatrooms, loot boxes), and have successfully attracted large numbers of investors. This paper attempts to discover the primary factors driving users to flock to these platforms. To answer this question, we have collected approximately a year worth of user data from one of the leading cryptocurrency derivative exchanges between 2020 and 2021. During that period, more than 7.5 million new user accounts were created on that platform. We build a regression analysis, accounting for the idiosyncrasies of the data at hand – notably, its non-stationarity and high correlation – and discover that prices of two major cryptocurrencies, Bitcoin and Ethereum, impact user registrations both in the short and long run. On the other hand, the influence of a less prominent coin, Ripple, and of a "meme" coin with a large social media presence, Dogecoin, is much more subtle. In particular, our regression model reveals the influence of Ripple prices vanishes when we include the SEC litigation against Ripple Labs, Inc. as an explanatory factor. Our regression analysis also suggests that the Chinese government statement regarding tightening cryptocurrency mining and trading regulations adversely impacted user registrations. These results indicate the strong influence of regulatory authorities on cryptocurrency investor behavior. We find cryptocurrency volatility impacts user registrations differently depending on the currency considered: volatility episodes in major cryptocurrencies immediately affect user registrations, whereas volatility of less prominent coins shows a delayed influence.

2012 ACM Subject Classification General and reference → Measurement; Applied computing → Digital cash

Keywords and phrases Cryptocurrency, Online Markets, Derivatives, Trading, Regression Analysis

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.8

Funding This research was partially supported by Ripple's University Blockchain Research Initiative (UBRI) at Carnegie Mellon and by the Carnegie Mellon CyLab Secure Blockchain Initiative. Some of the authors hold non-negligible cryptocurrency positions, but none on the platform under study.
Daisuke Kawai: DK is supported by the Japanese Government Long-Term Overseas Fellowship Program.

1 Introduction

Cryptocurrencies have had a growing impact on global finance. Shortly after the emergence of Bitcoin [33], use cases were primarily as a payment instrument for online fringe activities such as gambling, or the purchase of illegal goods [11, 31]. However, spot prices (i.e., the exchange rate to fiat currencies) rapidly skyrocketed – Bitcoin went from being worth nothing in 2009 to exceeding $60,000 in 2021 – so that cryptocurrencies became an important type of (speculative) financial asset [16].

Consequently, trading infrastructure rapidly expanded from spot exchanges, where people exchange cryptocurrencies for fiat currencies [32], to cryptocurrency derivative platforms [44]. Today, approximately 50–100 billion US dollars are traded every day on these off-chain derivative exchanges.[1] This number far exceeds that of cryptocurrency spot markets, and can be compared to the roughly 200 billion USD traded on the NASDAQ on a given day at the time of writing.[2] In short, cryptocurrency derivative markets are critical to understand the impact of cryptocurrencies on global finance.

The rapid increase in trading volume and user participation led financial regulators to pay close attention. The U.S. Securities and Exchange Commission (SEC) Chair famously emphasized the need for stronger regulations for better investor protection and market integrity [49]. At the international level, the Financial Stability Board (FSB) raised its risk evaluation of cryptocurrency and prioritized the risk assessment of cryptocurrency markets for 2022 [16, 17]. Out of these concerns about potential threats, financial authorities took regulatory measures regarding the cryptocurrency industry [5, 21, 50].

These regulatory changes, as well as large price swings, are expected to impact investor behavior. However, little quantitative analysis has been conducted to measure the degree of influence of all of these potential factors. The core contribution of this paper is to examine the degree to which price appreciation, volatility, and regulatory measures influence user decisions to engage in cryptocurrency investments. To do so, we rely on a dataset we obtained about the hourly performance data of more than eight million investors (registered by July 20, 2021, and most of whom are presumed to be invididual investors) in one of the largest cryptocurrency derivatives markets, from which we can derive how many new investors sign up to the exchange. We use that data to investigate how cryptocurrency prices affect the number of investors in the market with a regression model that can address the long-run relationship between the new registration and major cryptocurrency prices.

A prevailing narrative is that short-term speculation motivates cryptocurrency investments [15, 29] – if so, investors should flock to investment platforms as market volatility increases. We look at the effect of four cryptocurrencies ("reserve" cryptocurrencies like Bitcoin, "meme" currencies like Dogecoin, etc.) prices and volatility on investor registrations, and build a regression to tease out factors that appear to matter. Building this regression presents a number of technical challenges we elaborate on, and our analysis ultimately shows a nuanced picture. The number of investors increases over time, with both price rise and volatility acting as a crucial effect on the rate of increase. However, not all currencies are equal: contrary to Bitcoin or Ethereum, whose price hike and high volatilities immediately affect user registration, Ripple and Dogecoin prices have much less impact on user registrations in the short term, and it takes longer time for the impact of their high volatility to materialize. Our regression also shows the significant influence of regulatory measures. Our analysis shows

[1] `https://coinalyze.net/futures-data/global-charts/`
[2] `https://www.nasdaqtrader.com/Trader.aspx?id=DailyMarketSummary`

that the SEC litigation against Ripple Labs, Inc. and its executives basically negated any positive effect of Ripple's price rise on user registrations. The same analysis also suggests that the statement by the Chinese government that it was tightening cryptocurrency regulation[3] also adversely affected user registrations.

2 Related work

Bitcoin is a digital asset maintained by cryptographic primitives and distributed ledger technology. All transactions are recorded on a public ledger ("blockchain") and verified by peers engaging in a cryptographic puzzle ("miners"). Originally proposed as a payment method independent of trusted third parties [33], Bitcoin's use cases during its first few years were fraught with controversy: Meiklejohn et al. [31] showed that one of the major outlets for Bitcoin transactions was Silk Road, a marketplace for (mostly) illegal goods [11]. Moore and Christin showed that Bitcoin exchanges, where people trade Bitcoin for national ("fiat") currencies, frequently failed, and sometimes absconded with their users' money [32].

Despite (or maybe thanks to) the negative publicity, Bitcoin price skyrocketed within a few years. Multiple pieces of literature tried to understand why. Kristoufek [27] showed a correlation between Bitcoin price and the volume of related online search queries. In addition, they found that increased interest in Bitcoin inflates its price, which leads to a bubble-like price movement. Ciaian et al. [12] showed that Bitcoin's attractiveness to investors is an important driver, along with other conventional economic determinants. Urquhart [45] showed that an increase in realized Bitcoin price volatility is correlated to a larger number of related online searches one day later.

More generally, researchers proposed theoretical foundations to integrate various price determinants that had been observed empirically [7, 13, 34, 35, 40, 43]. Network effects appear critical: cryptocurrency appeal, and thus price, grows with the number of users, due to the increased security and (indirectly) usability a large user base provides. For instance, Liu and Tsyvinski's recent empirical analysis [28] shows that cryptocurrency prices correlate with the growth in the number of active on-chain addresses.

By analyzing conditional exposure to tail risks in other cryptocurrencies and in conventional financial asset prices, Borri [8] had showed cryptocurrency prices were affected by other cryptocurrencies, but were decoupled from conventional financial assets prices. Iyer [22] argues this may no longer be the case: correlation between cryptocurrency prices and conventional financial asset prices has been growing.

While this growing body of literature looks into correlations between cryptocurrencies and other financial assets, relatively little is known about market participants. Baur et al. [6] analyzed early Bitcoin holder demographics between 2011 and 2013 and showed that the main purpose of holding Bitcoin is for investment. By analyzing the BitMEX platform, Soska et al. [44] showed derivative investors were a mix of hobbyists and professional traders – with the latter often winning against the former. Kawai et al. [26] show that some derivatives investors provide unreliable investment advice on Twitter.

Despite these advances, many critical issues to characterize cryptocurrency investor behavior are yet to be answered. One of the issues is the influence of the price of major cryptocurrencies on potential investors – i.e., people who have not yet opened investment accounts in cryptocurrency markets, but are interested in investing. We argue this understanding is critical to better constructing a sustainable cryptocurrency investment environment.

[3] `https://www.gov.cn/xinwen/2021-05/21/content_5610192.htm?ivk_sa=1023197a`

3 Dataset

We obtained investor performance records over two years and a half from a large cryptocurrency derivative exchange public API, and use a subset of this data in the present paper. This section first briefly describes *perpetual futures*, the derivatives product predominantly traded on the exchange, before discussing the investor data present in our dataset.

3.1 Cryptocurrency derivative exchanges

While several platforms investigated various types of cryptocurrency contracts, BitMEX is generally credited with pioneering cryptocurrency derivative products, starting in November 2014 [2,3]. Compared to conventional derivatives markets, the most popular contract available is the *perpetual futures* contract, which, contrary to conventional derivative products (e.g., options), has *no expiry date*: Investors can hold their positions as long as their margin size is large enough to avoid liquidation. Soska et al. present a comprehensive study of BitMEX and of the perpetual futures contract [44]. Below we provide a quick summary of this type of contract, which subsequently became highly popular on all derivative exchanges, including the one we study in this paper.

3.1.1 Perpetual futures

Perpetual futures are investments in the future value of underlying cryptocurrencies: a typical case is the value of Bitcoin (BTC) against US dollar (USD) – or a related "stablecoin" (a cryptocurrency pegged to a fiat currency) like Tether (USDT). Investors of perpetual futures can go "long" or "short." An investor expecting a rise in BTC value against USD will go long (i.e., bet on the appreciation of BTC); conversely, investors expecting a decline will go "short." Longs and shorts are evenly matched among investors: every long contract is paired with a corresponding short contract placed by other investors.

Perpetual cryptocurrency future markets typically allow very high leverage, far beyond what their traditional finance counterparts tolerate. For instance, BitMEX [44] allowed up to 100x leverage. The platform we study allowed up to 125x leverage during the period we investigate (September 2020–July 2021). In short, an investor could invest up to 125 BTC worth of USD with only 1 BTC worth of USD as collateral. If the investor goes long (resp. short), and the value of bitcoin appreciates (resp. depreciates) against the US dollar, the investor can reap significant profit. On the other hand, leveraged positions are incredibly risky: for a 125x leveraged position, a swing of (slightly less than)[4] 0.8% compared to the purchase price, in the direction opposed to the bet made, results in liquidation. That is, the investor's position is immediately closed, and the investor loses all their money.

3.1.2 Performance indices

The exchange we study uses two indices to characterize investor performance: *Profit and Loss* (PnL) and *Return on Investment* (RoI). PnL shows the absolute profit (resp. loss) of an investment portfolio. An absolute metric, PnL tends to get large with investors who can take larger positions. On the other hand, the RoI, defined as the PnL divided by the investors' margin size (i.e., the funds the investor deposited in the market), is independent of the initial endowment.

[4] Due to transaction fees and other early liquidation mechanisms.

3.1.3 Rankings

The market we study provides ranking information of investors based on their PnL and RoI. The investor with the highest PnL (or RoI) ranks first, and other investors are sorted in descending order. Crucially, this ranking includes inactive investors who registered on the market but do not have any positions. These inactive investors have, by definition, a PnL and a RoI of zero, which is higher than that of investors who have incurred losses. As a result, the rank of an investor with a slightly negative PnL/RoI is orders of magnitude larger than that of an investor with a slightly positive PnL/RoI.

3.1.4 Cryptocurrency prices

The exchange also provides real-time prices of major cryptocurrencies via its public API. We collect these prices every minute throughout our measurement period. All collected prices are denominated in Tether (USDT).

3.2 Data collected

The cryptocurrency derivatives exchange we study started to publish ranking information on a leaderboard in mid-2020. While the leaderboard web front-end only shows the top investors, the public API initially provided information on every investor on the platform. Ranking data was updated hourly until May 9, 2021. Updates then shifted to a daily basis, until July 26, 2021. At that point, the exchange stopped providing ranking data for all investors; instead, the API now merely matches what the web front-end shows. As a result, we use data collected between August 20, 2020 and July 20, 2021.

4 Estimating the number of investors

4.1 Number of investors

As discussed above, the exchange API provides performance indices and ranking data about all investors. Unfortunately, to query data about a specific investor, we need their ID, and we cannot directly obtain the number of investors on the platform. Instead, we use ranking data as a proxy to estimate it.

Figure 1 shows the number of investors in our dataset, the maximum PnL rank among the investors, and their ratio at the beginning of each month in our observation period.

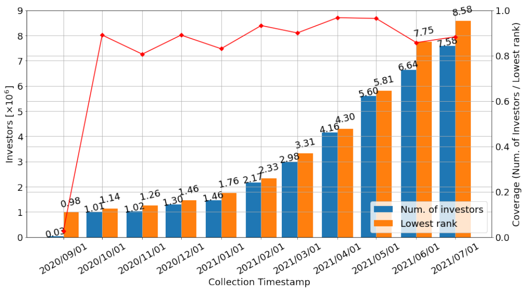


Figure 1 The number of investors in our dataset and the maximum (lowest) rank among the investors.

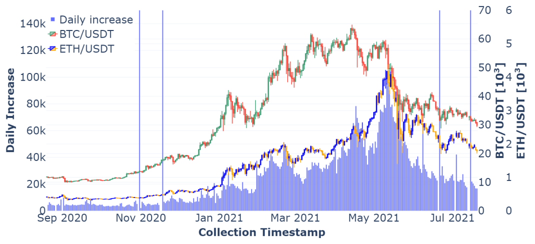


Figure 2 The daily increase in the number of investors in the market and the prices of Bitcoin (BTC) and Ether (ETH).

The figure shows that we collected data on more than one million investors and this ratio stays above 0.80 after October 2020 – the first month is an anomaly due to our data covering only a week or so. The large sample size ensures the lowest rank among collected investors is statistically very close to the number of investors in the market.[5] With this in mind, Figure 1 shows 7.5 million new investors joined the market increased in the ten months between September 1, 2020 to July 1, 2021.

Using maximum PnL rank as a proxy, we can estimate the number of investors in the market on a daily basis, even with an imperfect coverage of investors. Figure 2 shows both the daily increase in users on the platform, and the Bitcoin (BTC) and Ethereum (ETH) spot prices. Graphically, there seems to be a strong correlation between the number of new users joining the market, and the price of these currencies. The outliers (abnormally large increases) in November 2020 and July 2021 come from data collection errors due to changes in the exchange API implementation and collector breakdown.

In Section 6, we refine this intuition with a complete regression analysis.

4.2 Leaderboard data idiosyncracies

We have to account for certain idiosyncracies in our data. We infer registration numbers from the leaderboard data, which we itself get from a public API. However, there may be some lag times between what the API returns (leaderboard data may not be faithfully updated in real-time), and actual numbers; this can have an impact on our regression analysis.

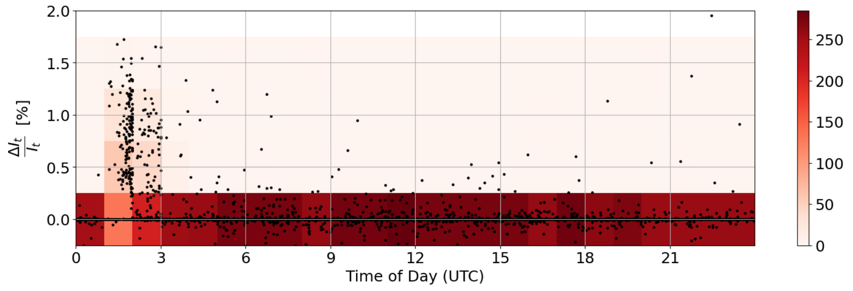


Figure 3 Hourly relative increase in the number of investors. The black dots show the exact time the largest rank in an hour was observed and the relative increase from the previous hour's largest rank. The background color shows the number of observations for a block of an hour and a 0.5% relative increase.

Figure 3 shows when new user registrations appear in our data, on a hourly basis. Each point corresponds to the relative increase in number of registered users compared to the previous hour, using the maximum leaderboard rank among observations in the hour as a proxy, as discussed earlier. We plot this data over our complete measurement interval (so, roughly 7,000 points corresponding to the number of hourly samples in our 10-month data). We observe that the reported number of users jumps during 0:00-3:00AM UTC on most days and usually does not change much thereafter. From this behavior, we hypothesize that the exchange updates the set of investors in the performance rankings once a day at midnight, integrating most, if not all, of those who registered in the previous day at that time.

[5] As a rough estimation, the probability that the relative error between the lowest rank and the (actual) number of investors in the market is equal to or less than 0.001% throughout our observation period (293 days) with one million samples ($\sim$ the number of investors at the beginning of October 2020) is: $Pr(\text{Relative Error} < 0.001\%) = \left(1 - (1 - 0.00001)^{1{,}000{,}000}\right)^{293} \simeq 0.987$. Given the increasing sample size, the actual probability is better than the approximation.

Therefore, we define the number of investors in the market in a day d as $I_d \equiv \max_{\tau \in d+1} I_\tau$, where I_τ is the largest observed leaderboard rank in a time slice τ. We also define the daily increase in a day d (N_d) as $N_d \equiv I_d - I_{d-1}$.

5 Regression analysis

We start by discussing the regression variables, before exploring how to construct our regression, considering the properties of the data we have at our disposal.

5.1 Variables

5.1.1 Daily user increase

Our first month of data has problematically sparse samples (3.0×10^4) and low coverage (2.67%). Hence, we discard it, and limit our analysis to October 1, 2020–July 20, 2021. We fix the handful of discontinuities observed in Figure 2 – due to data collection errors – by removing the outliers and replacing them with linear interpolations.

As Figure 2 shows, the daily increase N_d does not converge or revert to a mean value. In fact, as we will see in Section 6, N_d is a non-stationary variable. Fortunately, the Box-Cox transformation [9, 52] allows us to include such variables in an autoregressive model like the one we consider, by instead using a transformed variable that satisfies certain properties.[6] In our case, the logarithm of the daily increase, $\log N_d$, satisfies these requirements.

5.1.2 Prices

As noted above, we gather per-minute cryptocurrency prices. For currency X, at day d, we thus collect a vector of prices $\mathbf{P_{X,d}} = \{P_{X,1}, \ldots, P_{X,1440}\}$ corresponding to the 1 440 minutes in a day. The realized daily volatility $\sigma_{X,d}$ is:

$$\sigma_{X,d} = \sqrt{\frac{1440}{|\mathbf{P_{X,d}}|} \sum_{\tau \in d, \tau > 1} \left(\log P_{X,\tau} - \log P_{X,\tau-1}\right)^2} ,$$

where $P_{X,\tau}$ is the price of cryptocurrency X measured at time τ in day d.

Here too we use a Box-Cox transformation, and consider the logarithm of the daily average prices, $\log \bar{P}_{X,d}$, as an explanatory variable. Its first difference $\Delta \log \bar{P}_{X,d} \equiv \log \bar{P}_{X,d} - \log \bar{P}_{X,d-1}$ is the logarithmic return of the price, showing the approximate percentage change in the daily price. We will also use the realized volatility $\sigma_{X,d}$ as an additional explanatory variable. To calculate daily average prices $\bar{P}_{X,d}$ in a manner robust to short-lived volatile price movements, we will follow Biais et al. [7], by calculating the average of median values over short time intervals (5 minutes).

We select four cryptocurrencies for their importance and/or unique characteristics.

Bitcoin (BTC). Bitcoin has the largest market cap among cryptocurrencies, and is frequently touted as the "reserve currency" of the cryptocurrency ecosystem. BTC-USDT is the most popular futures contract in the exchange we consider, and Bitcoin presents the largest open interest, that is, the total amount (in USDT) of futures contracts held by market participants.

[6] Namely, that the mean and variance of its first difference are stationary.

Ethereum (ETH). Ethereum has the second largest market cap among cryptocurrencies, and features the second largest open interest in the exchange. ETH is the utility token in the Ethereum blockchain, which supports many smart contracts, including the majority of decentralized finance (DeFi) contracts and protocols. ETH thus gives us some insights into potential investor interests (and beliefs) in more elaborate blockchain proposals.

Ripple (XRP). XRP is another major cryptocurrency with a decentralized consensus mechanism [10]. Ripple Labs, Inc., the company behind XRP, was sued by the U.S. Securities and Exchange Commission (SEC) in December 2020.[7] At the time of writing, the suit has not been resolved. Among all cryptocurrency legal wranglings, this case is interesting to understand the potential influence of regulatory measures on user interest in a pretty popular coin, specifically, the third largest coin by market capitalization at the time.[8] Hence, XRP could give us insight into investor reactions to regulatory issues.

Dogecoin (DOGE). Originally a "meme" cryptocurrency primarily designed with humorous goals in mind, DOGE received increased attention due to numerous social media campaigns by influencers touting its potential (notably for tips and micropayments). As a result of the attention, DOGE soared in value from 0.005 USDT in January 2021 to 0.5 USDT in May 2021, before hitting an all-time high of 0.75 USDT on May 7, 2021. Social media attention faded away shortly thereafter, and the currency lost significant value. DOGE is thus an interesting currency to include, as a loose proxy for social media activity.

Table 1 summarizes statistics for the logarithms and realized volatilities for the four cryptocurrencies above. Reflecting the price hike in DOGE in early 2021, the standard deviations for DOGE are higher than other variables. We will later use the mean values and standard deviations of level variables for the Principal Component Analysis (PCA). Appendix A shows the plot of daily average prices and realized volatilities of four selected cryptocurrencies.

Table 1 Descriptive statistics for the daily increase in the number of investors and the logarithm of daily average price and realized volatility of BTC, ETH, XRP, and DOGE.

	$\log N$	$\log \bar{P}_{BTC}$	$\log \bar{P}_{ETH}$	$\log \bar{P}_{XRP}$	$\log \bar{P}_{DOGE}$	σ_{BTC}	σ_{ETH}	σ_{XRP}	σ_{DOGE}
• **Level variable**									
Mean	9.862	10.371	7.145	-0.636	-3.396	0.047	0.059	0.087	0.100
Median	10.077	10.469	7.434	-0.610	-2.917	0.042	0.051	0.071	0.066
Std. Dev.	0.899	0.518	0.714	0.583	1.923	0.025	0.038	0.060	0.099
Max.	11.465	11.059	8.346	0.589	-0.370	0.233	0.475	0.417	0.900
Min.	7.654	9.261	5.827	-1.556	-5.990	0.010	0.019	0.018	0.015
• **First difference**									
Mean	0.006	0.004	0.006	0.003	0.014				
Median	-0.012	0.006	0.006	0.002	0.001				
Std. Dev.	0.212	0.037	0.046	0.077	0.127				
Max.	1.174	0.126	0.163	0.291	1.238				
Min.	-1.256	-0.145	-0.196	-0.321	-0.470				

[7] See `https://www.sec.gov/news/press-release/2020-338`.
[8] See `https://coinmarketcap.com/historical/20201220/`

5.2 Method

All of our variables are time-dependent and potentially highly correlated. An unbiased regression analysis generally requires time-dependent variables to be at least (weak-)stationary [4, 19, 20, 30] and to present low correlation [39]. Stationarity means the mean values should be finite, time-invariant, and auto-covariances should only depend on the time interval over which they are calculated. By successively differencing a non-stationary variable, we might eventually end up with a stationary variable (e.g., a random walk variable y_t following $y_t = y_{t-1} + \epsilon_t$ with white noise ϵ_t is not stationary, but its first difference, $\Delta y_t = \epsilon_t$, is). We denote by $I(d)$ the number of successive differencing operations required to make the tested variable stationary. $I(d)$, also called the *order of integration*, will be key in determining which regression model to use. Also, keeping the correlation between explanatory variables low is an essential part of pre-processing to hold a regression analysis informative.

5.2.1 Unit root test

To check stationarity, we rely on the *unit root test* technique. One of the best known such tests is the Augmented Dickey-Fuller (ADF) test [14]. ADF tests the null hypothesis that the variable tested is a unit root (i.e., $I(1)$). If it rejects the null hypothesis with a small enough p-value, the process is deemed stationary ($I(0)$). The Phillips-Perron (PP) test [38] is also widely used to test stationarity. PP assumes the same null hypothesis as ADF, but allows heteroskedasticity and autocorrelation in the error term. We will use both PP and ADF in our analysis.

5.2.2 Principal Component Analysis

We employ *Principal Component Analysis* (PCA, [18]) to solve the problem of high correlation between explanatory variables. PCA is an orthogonal projection of the original variables (X) onto a lower-dimensional set of variables (S_L), preserving as much information as possible: $S_L = \widehat{X} W_L$, where L is the dimension of PCA-vector space ($L \leq \dim(X)$). W_L is the coefficient matrix for constructing principal components from normalized price-related variables $\widehat{X_i}$, which is composed of the variables normalized with its mean value ($\overline{X_i}$) and standard deviation ($\sqrt{Var(X_i)}$): $\widehat{X_i} \equiv \frac{X_i - \overline{X_i}}{\sqrt{Var(X_i)}}$. Because PCA components are orthogonal, PCA prevents the regression analysis from being contaminated by highly correlated components. We can then calculate the original variables' coefficients from those for PCA components by simple linear algebraic manipulations.

5.2.3 Autoregressive distributed lag model

We will build our regression using an autoregressive distributed lag (ARDL) model, which, contrary to most regression models, can accommodate a mixture of $I(0)$ variables and $I(1)$ variables [36]. This makes it particularly suited to our problem, given the apparent non-stationarity of at least some of our variables.

We will use the following unrestricted error correction model (UECM) representation of ARDL in our analysis:

$$\begin{aligned}\Delta\widehat{\log N_d} = \; & c_0 + \textstyle\sum_S \gamma_S I_{S,d} \\ & + \pi_0 \widehat{\log N_{d-1}} + \textstyle\sum_i \pi_i v_{i,d-1} + \sum_i \pi'_i w_{i,d-1} \\ & + \textstyle\sum_{i=1}^{p-1} \alpha_i \Delta \widehat{\log N_{d-i}} \\ & + \textstyle\sum_i \sum_{j=0}^{q_i-1} \beta_{i,j} \Delta v_{i,d-j} + \sum_i \sum_{j=0}^{q'_i-1} \beta'_{i,j} \Delta w_{i,d-j} \\ & + \epsilon_d \;,\end{aligned} \tag{1}$$

where p, $q_i(q_i')$, $I_{S,d}$, and ϵ_d represent the lag order of the normalized daily increase $\left(\widehat{\log N_d} \equiv \frac{\log N_d - \overline{\log N_d}}{\sqrt{Var(\log N_d)}}\right)$, those for principal components for daily average prices (v_i) and realized volatilities (w_i) in d-th day, indicator variables of interest (labeled by S), and the error term, respectively. α, β, β', γ, π_0, π, and π' are regression coefficients.

Pesaran et al. [36] propose a bounds test in an ARDL model (*PSS-bounds test*), to determine the existence of a long-run equilibrium relationship (i.e., cointegration) between variables. The test compares the test statistic with two critical boundaries. If the tested statistic is larger than the upper boundary (called $I(1)$-boundary), the test confirms the existence of a long-run relationship; On the other hand, if the tested statistic is lower than the lower boundary ($I(0)$-boundary), the test rejects the existence of a long-run relationship. If the tested statistic falls between the $I(0)$ and $I(1)$ boundary, no conclusion about the existence, or lack thereof, of a long-run relationship can be derived. PSS-bounds test has five cases (Case I-V) for the specification of deterministic terms. We consider Case I (no constant term in the ARDL model), Case II (a constant term in the ARDL model and cointegration), and III (a constant term in the ARDL model, but no constant term in cointegration). In the UECM representation, the cointegrations are mainly given by the second line in Eqn. (1):

$$\begin{aligned}
\widehat{\log N_d} + \tfrac{1}{\pi_0}\left(\textstyle\sum_i \pi_i v_{i,d} + \sum_j \pi_j' w_{i,d}\right) &= 0 \quad \text{(Case I)}\ , \\
\widehat{\log N_d} + \tfrac{1}{\pi_0}\left(\mu + \textstyle\sum_i \pi_i v_{i,d} + \sum_j \pi_j' w_{i,d}\right) &= 0 \quad \text{(Case II)}\ , \\
\widehat{\log N_d} + \tfrac{1}{\pi_0}\left(\textstyle\sum_i \pi_i v_{i,d} + \sum_j \pi_j' w_{i,d}\right) &= 0 \quad \text{(Case III)}\ ,
\end{aligned} \tag{2}$$

where μ is the deterministic term(s) for cointegration.

Intuitively, Eqn. (1) says that the change in $\widehat{\log N_d}$ is explained by (1) the short-run change in itself and explanatory variables and (2) the deviation from cointegration (i.e., long-run equilibrium status) if it exists.

We can consider the marginal effect of explanatory variables ($\frac{\partial \widehat{\log N_{d+k}}}{\partial V_{X,d}}$) in an arbitrary temporal duration k (≥ 0) when they converge to zero over time.

Short-run multipliers ($\frac{\partial \widehat{\log N_d}}{\partial V_{X,d}}$) represents the immediate impact of an explanatory variable $V_{X,d}$. In Eqn. (1), short-run multipliers are given by $\beta_{i,0}$ and $\beta_{i,0}'$. The cumulative marginal effect up to k-th day ($\sum_{l=0}^{k} \frac{\partial \widehat{\log N_{d+k}}}{\partial V_{X,d+l}}$) shows the accumulated impact of change in explanatory variables lasting for k days, and converges to a finite value as k increase when the marginal effect converges to zero. Since $\sum_{l=0}^{k} \frac{\partial \widehat{\log N_{d+k}}}{\partial V_{X,d+l}} = \sum_{l=0}^{k} \frac{\partial \widehat{\log N_{d+l}}}{\partial V_{X,d}}$, we can also interpret this quantity as the cumulative effect that today's change in an explanatory variable will cause for k days in the future.

Long-run multipliers ($\lim_{k\to\infty} \sum_{l=0}^{k} \frac{\widehat{\log N_{d+k}}}{\partial V_{X,d+l}}$) denote the cumulative marginal effect on $\widehat{\log N_{d+k}}$ coming from a persistent change in an explanatory variable. From the discussion above, this quantity represents the cumulative effect today's change in an explanatory variable causes in the long future. Going back to Eqn. (1), long-run multipliers are given by $\frac{-\pi}{\pi_0}$ and $\frac{-\pi'}{\pi_0}$ for the principal components of daily average prices and realized volatilities, respectively.

Our analysis considers two major regulatory measures that affected cryptocurrency prices in our observation period, using two indicator variables (I_S): (1) the influence of the SEC litigation against Ripple Labs, Inc. and (2) the Chinese government's statement that it planned to tighten cryptocurrency regulation. The big swings in XRP price after the announcement of the lawsuit may affect newly-participating investor behavior. To capture the potential effects, we introduce the indicator variable:

$$I_{SEC,d} = \begin{cases} 1 & d \text{ is before Dec. 22, 2020 }, \\ 0 & \text{otherwise }. \end{cases} \tag{3}$$

Dec. 22, 2020 is the day the SEC publicly announced the lawsuit. In the definition of Eqn. (3), the sum of the constant term and I_{SEC} $(c_0 + I_{SEC})$ reresents the constant percentage change in user registrations before the lawsuit was announced; this becomes a constant term (c_0) after that announcement. We employ this definition of I_{SEC} to avoid shifting the critical values of the PSS-bounds test [36].[9]

We use another indicator variable to capture the effect of the Chinese government's statement. It was published on May 21, 2021 [5]. This statement is considered to have had a significant impact on wide range of cryptocurrencies adversely.

$$I_{CHN,d} = \begin{cases} 1 & d \geq \text{May 21, 2021 }, \\ 0 & \text{otherwise }. \end{cases} \tag{4}$$

A statistically significant coefficient for I_{SEC} and I_{CHN} would indicate a spill-over effect that is not absorbed in cryptocurrency prices. Geofencing has been an issue for major crypto-exchanges as evidenced by multiple legal proceedings [46–48, 51], with investors allegedly residing in countries that restrict participation (specifically, the US and China). We have no reason to believe the market we study is immune to geofencing issues. Hence, we expect the announcement of these regulatory actions to impact potential investor behavior. Moreover, these measures were announced within our observation period, making it possible to precisely gauge their impact. We also considered the UK ban on retail crypto-derivatives trading that became effective on Jan. 6, 2021[10] as a potentially relevant case, but did not observe any significant impact. We cannot distinguish whether this is because the announcement was made before our observation period started (June 10, 2020), and investors had already factored it into account, or because UK regulations have less of an overall impact.

Our analysis uses *urca* package for R [37] for unit-root tests and *statsmodels* package for Python [42] for the remaining analyses. We employ heteroskedasticity autocorrelation (HAC) robust variance estimation throughout our analyses to compensate for the potential impact of determinants other than our selected terms and autocorrelation.

6 Results

We start with unit root tests to ensure all variables are $I(0)$ or $I(1)$ so that we can use ARDL. Then, we consider the correlation between explanatory variables and finally perform a complete analysis of our ARDL model to tease out the factors behind user registrations.

6.1 Unit root test

Table 2 summarizes the unit root test results for level variables and their first difference, where we determine the lag order in ADF to minimize Akaike Information Criterion (AIC) [1]. Both ADF and PP provide consistent results about variable stationarity. The analysis shows that (taking their logarithms), the daily user registration increases and the daily average

[9] PSS-bounds test's critical values must be modified if the regression formula includes indicator variable(s) that do not disappear as the observation period increases. We defined I_{SEC} in Eqn. (3) to mitigate the potential contamination from long-lasting non-zero indicator variables.

[10] `https://www.fca.org.uk/news/press-releases/fca-bans-sale-crypto-derivatives-retail-consumers`

Table 2 Unit root test results.

	Level variable		First difference		
Variable	Intercept	Intercept and Trend term	Intercept	Intercept and Trend term	Order of integration
ADF test					
$\log N$	-1.91	-0.49	-10.07^{***}	-10.31^{***}	$I(1)$
$\log \bar{P}_{BTC}$	-2.38	-0.45	-6.53^{***}	-7.23^{***}	$I(1)$
$\log \bar{P}_{ETH}$	-1.84	-0.60	-10.52^{***}	-10.70^{***}	$I(1)$
$\log \bar{P}_{XRP}$	-1.71	-1.55	-11.12^{***}	-11.16^{***}	$I(1)$
$\log \bar{P}_{DOGE}$	-1.10	-1.39	-7.85^{***}	-7.87^{***}	$I(1)$
σ_{BTC}	-4.04^{***}	-4.02^{***}	-9.40^{***}	-9.41^{***}	$I(0)$
σ_{ETH}	-4.11^{***}	-4.17^{***}	-9.78^{***}	-9.78^{***}	$I(0)$
σ_{XRP}	-5.31^{***}	-5.29^{***}	-7.70^{***}	-7.71^{***}	$I(0)$
σ_{DOGE}	-6.44^{***}	-6.53^{***}	-9.67^{***}	-9.66^{***}	$I(0)$
PP test					
$\log N$	-1.82	-1.98	-26.53^{***}	-26.93^{***}	$I(1)$
$\log \bar{P}_{BTC}$	-2.39	-0.27	-14.25^{***}	-14.65^{***}	$I(1)$
$\log \bar{P}_{ETH}$	-1.85	-0.44	-13.03^{***}	-13.15^{***}	$I(1)$
$\log \bar{P}_{XRP}$	-1.72	-1.57	-13.03^{***}	-13.04^{***}	$I(1)$
$\log \bar{P}_{DOGE}$	-1.06	-1.29	-14.60^{***}	-14.60^{***}	$I(1)$
σ_{BTC}	-8.18^{***}	-8.40^{***}	-31.31^{***}	-31.30^{***}	$I(0)$
σ_{ETH}	-9.47^{***}	-9.77^{***}	-33.92^{***}	-33.88^{***}	$I(0)$
σ_{XRP}	-7.69^{***}	-7.70^{***}	-26.83^{***}	-26.80^{***}	$I(0)$
σ_{DOGE}	-6.80^{***}	-6.91^{***}	-21.84^{***}	-21.80^{***}	$I(0)$

*, **, *** denote significance at the 10%, 5%, and 1% level, respectively.

prices are unit root $I(1)$, but volatilities are stationary, i.e., $I(0)$. Hence, we can use ARDL in our analysis for daily registrations and PCA components constructed from price-related variables: the PCA components, which are composed of the linear combination of $\log \bar{P}_X$ and σ_X, are at most $I(1)$.

6.2 Principal Component Analysis

Table 3 shows that the Pearson correlation coefficients between log daily average prices and realized volatilities are so high that regression analysis with these variables will suffer from a multi-collinearity problem [18, 39].

Table 3 Pearson correlation coefficients for daily average prices and realized volatilities.

	Daily average price				Realized volatility			
	$\log \bar{P}_{BTC}$	$\log \bar{P}_{ETH}$	$\log \bar{P}_{XRP}$	$\log \bar{P}_{DOGE}$	σ_{BTC}	σ_{ETH}	σ_{XRP}	σ_{DOGE}
$\log \bar{P}_{BTC}$	1.00	0.90	0.64	0.78	0.31	0.25	0.27	0.33
$\log \bar{P}_{ETH}$		1.00	0.78	0.95	0.31	0.30	0.20	0.31
$\log \bar{P}_{XRP}$			1.00	0.81	0.12	0.18	0.25	0.24
$\log \bar{P}_{DOGE}$				1.00	0.25	0.25	0.13	0.29
σ_{BTC}					1.00	0.92	0.56	0.52
σ_{ETH}						1.00	0.57	0.49
σ_{XRP}							1.00	0.54
σ_{DOGE}								1.00

Therefore, we consider the Principal Component Analysis (PCA) of daily average prices and realized volatilities. Table 4 summarizes the construction of PCA components from normalized log daily average prices ($\widehat{\log \bar{P}_X}$) and realized volatilities ($\widehat{\sigma_X}$). The table shows

Table 4 Principal component coefficients and percentage of variance explained by each principal component.

	Daily average price					Realized volatility				
	log $\widehat{\bar{P}_{BTC}}$	log $\widehat{\bar{P}_{ETH}}$	log $\widehat{\bar{P}_{XRP}}$	log $\widehat{\bar{P}_{DOGE}}$	% of variance	$\widehat{\sigma_{BTC}}$	$\widehat{\sigma_{ETH}}$	$\widehat{\sigma_{XRP}}$	$\widehat{\sigma_{DOGE}}$	% of variance
PC1	0.015	0.017	0.015	0.016	86.0	0.019	0.019	0.016	0.015	70.4
PC2	-0.060	-0.018	0.071	0.011	9.4	-0.031	-0.033	0.027	0.050	16.3
PC3	-0.074	0.036	-0.069	0.095	4.1	-0.013	-0.007	0.070	-0.050	11.3
PC4	-0.146	0.368	0.008	-0.246	0.4	-0.145	0.144	-0.007	0.009	2.0

that the first component (PC1) for both log daily average prices (v_1) and realized volatilities (w_1) are composed of the almost equally weighted sum of four coins, which basically denotes the *average* trend of cryptocurrency prices and volatilities. The second component in daily average prices (PC2) has large BTC and XRP coefficients with opposite signs, capturing how XRP price trends deviate (or get "decoupled") from BTC price trends, to which the SEC litigation against Ripple may have contributed. The realized volatilities' second component measures the volatility difference between major coins (BTC and ETH), on the one hand, and relatively less prominent coins (XRP and DOGE), on the other hand. Figure 4 shows

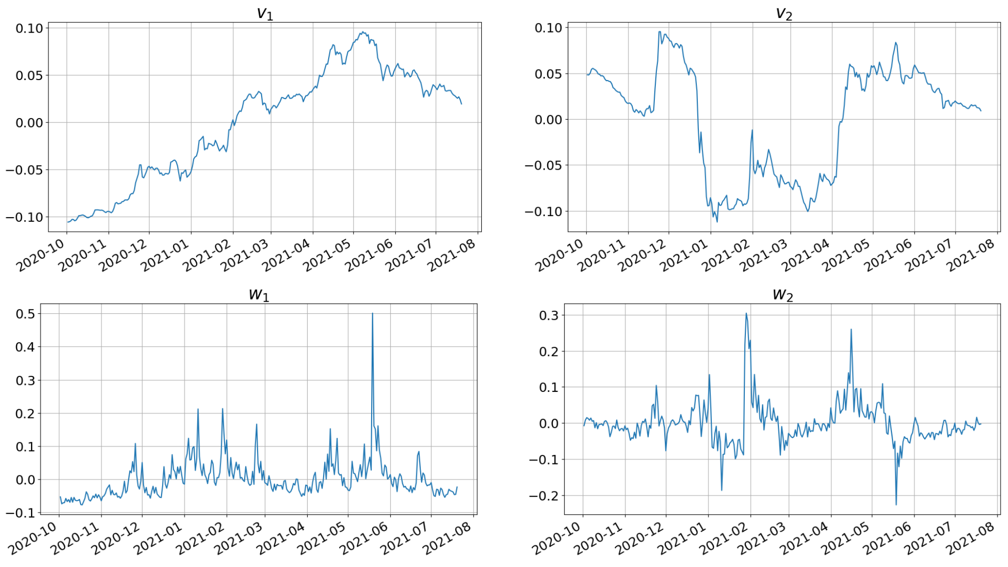


Figure 4 First and second components for log daily average prices (top) and realized volatilities (bottom).

the first and second components for log daily average prices (v_1, v_2) and realized volatilities (w_1, w_2). As we expect from Table 4, the first component for log daily average prices (v_1) represents cryptocurrency price trends: rising until May 2021 and the subsequent downturn. The second component for log daily average (v_2) prices denotes a sudden decrease in the value in late December 2020, when the SEC announced its litigation against Ripple. The increase in early April 2021 might be caused by investors getting more relaxed about the impact of this litigation on XRP [23]. Finally, the second component for realized volatilities (w_2) displays sharp positive spikes in February and April 2021 caused by XRP and DOGE as well as the negative spike in May 2021 due to the high volatility of BTC.

Table 5 Model selection for ARDL analysis.

	Num. of PCA components for log daily average prices (Cum. % of variance)	**Num. of PCA components for realized volatilities** (Cum. % of variance)	**Indicator variables**
Model 1	1 (86.0%)	1 (70.3%)	No
Model 2	2 (95.4%)	1 (70.3%)	No
Model 3	2 (95.4%)	2 (86.6%)	No
Model 4	2 (95.4%)	2 (86.6%)	Chinese govt. statement
Model 5	2 (95.4%)	2 (86.6%)	Chinese govt. statement + SEC XRP lawsuit

6.3 ARDL model analysis

We next delve into our regression analysis with the ARDL model. We determine the lag order of autoregressive terms ($\Delta\widehat{\log N_d}$) and distributed lag terms ($\Delta v_{i,d}$ and $\Delta w_{i,d}$) to minimize the Bayesian information criterion (BIC) [41]. In determining the lag orders, we limit ourselves to a maximum lag order of ten for both autoregressive terms and distributed lag terms. This means that we consider a lag of up to ten days. Then, we select a model with the smallest BIC from those with lag orders higher than or equal to one for all distributed lag terms, so that we can construct a UECM representation. Fortunately, models with smaller lags yield smaller BIC values than those with higher orders, so our self-imposed limitation for the maximum lag order does not affect our results.

Model Specification. We consider five models, summarized in Table 5, for analyzing the influence of cryptocurrency prices on user registrations to a cryptocurrency derivatives market. Models 1–3 analyze the effect of model complexity. Models 4 and 5 measure the influence of regulatory measures on daily registration by comparing them with Model 3. During model selection, we found that models with different combinations of principal components all reduced to those listed in Table 5. For example, a model selection starting from a model with the first principal component for log daily prices and the first and second components for realized volatilities reduces to Model 1 in optimization.

6.3.1 Fitting result

Table 6 summarizes the estimation results for all ARDL regressions. First, our full-fledged Model 5 exhibits minimum values for all information criteria. That indicates Model 5 is the best among fitted models. Model 4 presents the second-smallest information criteria values. These results indicate that adding the indicator variables for controlling regulatory measures, as well as the selection of principal components, enhances the explanatory power of our ARDL models.

Second, the first difference of the first principal component (PC1) for the logarithm of the daily average price ($\Delta v_{1,d}$) significantly influences user registrations. Given the standard deviation for the daily registration ($\log N$) and the logarithm of daily average prices ($\log \bar{P}_X$) in Table 1 and the coefficients for PC1 in Table 4, a 1.0% increase in cryptocurrency prices for a given day will roughly drive a 2.0% increase in user registrations in the same day.[11] This pattern consistently shows up in all models, which indicates that rising cryptocurrency prices positively correlate with decisions of potential investors to join the market.

[11] Due to normalization while constructing the PCA components, we have to multiply the ratio of standard deviations ($\sqrt{Var(\log N)/Var(\log P_X)}$) and the coefficient for constructing PCA (Table 4) to the coefficient for ARDL in Table 6 to get the coefficient in their original scales.

Table 6 ARDL regression results for Models 1–5. The values in parentheses are standard errors.

	Model 1	**Model 2**	**Model 3**	**Model 4**	**Model 5**
Const. (c_0)	-0.002	-0.005	-0.004	0.038^{**}	-0.041
	(0.010)	(0.010)	(0.010)	(0.015)	(0.034)
$\log \widehat{N_{d-1}}$	-0.281^{***}	-0.440^{***}	-0.477^{***}	-0.707^{***}	-0.712^{***}
	(0.073)	(0.096)	(0.099)	(0.104)	(0.099)
$\Delta \log \widehat{N_{d-1}}$	-0.234^{**}	-0.152^{*}	-0.145^{*}	–	–
	(0.095)	(0.089)	(0.090)		
Log daily average price					
$v_{1,d-1}$	4.193^{***}	6.749^{***}	7.268^{***}	11.384^{***}	12.730^{***}
	(1.098)	(1.474)	(1.517)	(1.734)	(1.695)
$v_{2,d-1}$	–	-1.050^{***}	-1.211^{***}	-1.401^{***}	-2.371^{***}
		(0.253)	(0.275)	(0.261)	(0.467)
$\Delta v_{1,d}$	22.940^{***}	26.536^{***}	24.339^{***}	23.245^{***}	24.116^{***}
	(2.506)	(2.596)	(3.265)	(2.958)	(2.916)
$\Delta v_{2,d}$	–	-4.764^{***}	-4.783^{***}	-5.091^{***}	-6.388^{***}
		(1.187)	(1.170)	(1.194)	(1.425)
Realized Volatility					
$w_{1,d-1}$	0.960^{***}	1.334^{***}	1.404^{***}	1.990^{***}	2.241^{***}
	(0.344)	(0.336)	(0.360)	(0.308)	(0.341)
$w_{2,d-1}$	–	–	0.588^{***}	0.343^{**}	0.488^{***}
			(0.211)	(0.174)	(0.213)
$\Delta w_{1,d}$	2.017^{***}	2.136^{***}	2.038^{***}	1.900^{***}	2.028^{***}
	(0.450)	(0.390)	(0.328)	(0.349)	(0.347)
$\Delta w_{1,d-1}$	0.793^{***}	0.638^{***}	0.594^{**}	–	–
	(0.220)	(0.216)	(0.235)		
$\Delta w_{2,d}$	–	–	0.138	0.343	0.177
			(0.303)	(0.174)	(0.301)
Indicator variables					
I_{CHN}	–	–	–	-0.201^{***}	-0.145^{***}
				(0.047)	(0.049)
I_{SEC}	–	–	–	–	0.239^{**}
					(0.095)
PSS-bounds test					
Case-I (w/o const.)	5.263^{***}	5.395^{***}	4.729^{***}	7.442^{***}	7.442^{***}
Case-II (w const.)	3.969^{**}	4.411^{***}	4.063^{**}	6.182^{**}	6.182^{***}
Case-III (w/o const.)	5.271^{**}	5.457^{***}	4.779^{***}	7.408^{***}	7.408^{***}
Best-fit model	UECM(2,1,2)	UECM(2,1,1,2)	UECM(2,1,1,2,1)	UECM(1,1,1,1,1)	UECM(1,1,1,1,1)
Num. of observations	292	292	292	292	292
Log-Likelihood	64.647	78.637	82.623	95.690	99.369
AIC	-111.294	-135.274	-139.246	-167.380	-172.737
BIC	-78.265	-94.905	-91.538	-123.300	-124.984
HQIC	-98.061	-119.100	-120.132	-149.721	-153.607
R^2	0.325	0.387	0.404	0.454	0.467

***, **, and * represent significance at the 1%, 5%, and 10% level.

Third, the first difference of the first principal component for the realized volatilities ($\Delta w_{1,d}$), i.e., the change in the overall volatility trend, shows a similar influence pattern. The same-day increase in the variable ($\Delta w_{1,d}$) consistently has a significant impact on the daily increase in the number of investors in all models. In the original variables scale, a 0.01 increase in all realized volatilities causes a 3.0% larger user registration on the same day.

These influence patterns of (the logarithm of) daily average prices and realized volatilities are consistent with often heard narratives about motivations for engaging in cryptocurrency investments: cryptocurrency investors are supposedly primarily driven by speculation, so cryptocurrency price rise and high volatilities will drive more user participation.

However, our regression analysis also shows a more complex picture of the factors influencing investor behavior. Model 4, which includes the indicator variable that captures the potential impact of the Chinese government's statement (I_{CHN}), suggests that the constant term (c_0) is positive and significant at the 5% level. This implies that the daily registration increases ($\log N$) by 3.4% every day in the original scale, which is given by multiplying I_{CHN} by the standard deviation of $\log N$ ($= 0.038 \times 0.899$), even if cryptocurrency prices were stable *before* the statement was published. However, our analysis shows that the Chinese government statement poured cold water on investor enthusiasm. Specifically, the influence of I_{CHN} term swallows the constant term, and the sum of these two terms ($c_0 + I_{CHN}$) turns to negative (-0.163), meaning that new registrations will decrease by 14.7% ($= 0.163 \times 0.899$) every day in the original scale if cryptocurrency prices are stable. This result evidences the strong impact of a specific regulatory issue on investor behavior that is not explained by decreasing cryptocurrency prices. Note that the constant term for Model 4 does not have to be zero, although we employ PCA for both the dependent and explanatory variables. This is because the indicator variable (I_{CHN}) is not centered.[12]

Finally, we consider the effect of the SEC litigation against Ripple on user registrations. The constant term (c_0) for Model 5 loses significance at the 5% level, and I_{SEC} holds a large coefficient of 0.239. So, the constant percentage change in user registrations before the lawsuit announcement ($c_0 + I_{SEC}$) is 0.198, suggesting a 17.8% daily increase in user registration in its original scale. However, this increase subsided after the litigation was announced, once again showing that a regulatory issue impacted user behavior.

PSS bounds test result. Next, we consider the long-run effect of prices in detail. Since marginal effects $\left(\frac{\partial \widehat{\log N_{d+k}}}{\partial V_{X,d}}\right)$ for all explanatory variables converge to zero as time goes on (see Appendix B), we can consider a stable long-run equilibrium state.

Since we use normalized variables for regression (see Section 5), the constant terms for Models 1–3 are theoretically zero, consistent with the results in Table 6. Hence, the appropriate bound test case specification for Models 1–3 is Case-I in Eqn. (2). On the other hand, Table 6 shows that the constant term for Model 4 is non-zero at the 1% significance level, indicating that Case-II or Case-III are appropriate. There is no theoretical restriction to determine the appropriate bound test case specification for Model 5, so we consider Case I–III.

Fortunately, all PSS bounds test results in Table 6 reject the null hypothesis that there is no cointegration (i.e., an equilibrium state) between the daily user registration and the price-related variables at the 5% significance level. This result strongly suggests the existence of a long-run equilibrium relationship between the inflow of new investors to the cryptocurrency investment market and cryptocurrency prices.

Figure 5 shows the observed user registration and the estimation from our cointegrations in Models 3–5. It demonstrates that our cointegration replicates the observed data well. This result has crucial implications. Since cryptocurrency derivatives are traded on off-chain exchanges, investor demographics, such as population, are not fully observable. This can cause considerable information asymmetry between market operators and outsiders, such as investors and financial regulators. However, our cointegration may be useful as an easy way to estimate the number of market investors from publicly available price data, thereby reducing this information discrepancy.

[12] A linear regression of a normalized dependent variable with normalized explanatory variables requires that the constant term be zero, as is the case in Models 1–3. However, the sum of the constant term and the average value of the indicator variable(s) has to be zero when the un-centered variable(s) is/are integrated. In Model 4, that sum is $0.038 - 0.201 \times \frac{61}{292} \simeq -4.0 \times 10^{-3}$, which satisfies this condition.

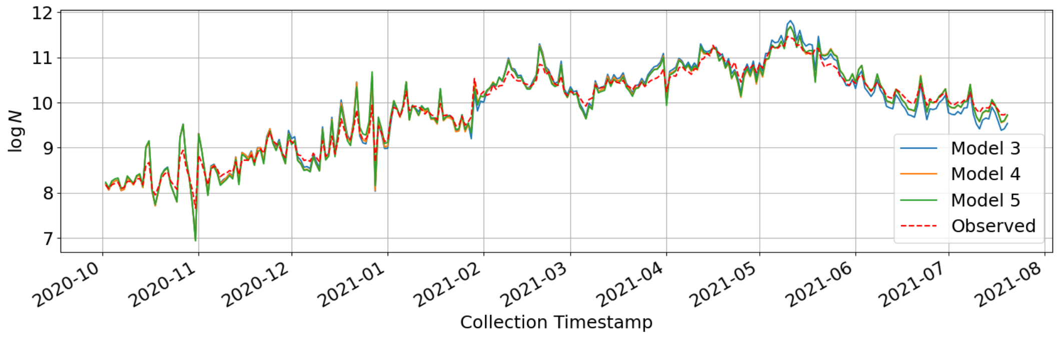


Figure 5 The logarithm of daily user registration increase replicated from our cointegration (solid lines) in Models 3–5 and its observed value (red dashed line).

6.3.2 Individual cryptocurrency influence

This section considers the influence of each cryptocurrency on daily user registrations. Since a linear algebraic relation connects the original price-related variables and principal components ($S_L = \widehat{X}W_L$), we can derive the coefficients for the daily average prices and realized volatilities in their original scale from those for principal components.

Table 7 summarizes the short-run and long-run multipliers for the daily average prices and realized volatilities in Models 3–5 in their original scales.

Table 7 Long-run and short-run multipliers. The value in the parentheses are standard errors for estimates.

	Model 3		**Model 4**		**Model 5**	
Multipliers	Short-run	Long-run	Short-run	Long-run	Short-run	Long-run
$\log \bar{P}_{BTC}$	1.145^{***}	0.668^{***}	1.148^{***}	0.633^{***}	1.307^{***}	0.822^{***}
	(0.181)	(0.035)	(0.176)	(0.028)	(0.194)	(0.082)
$\log \bar{P}_{ETH}$	0.621^{***}	0.378^{***}	0.605^{***}	0.384^{***}	0.653^{***}	0.452^{***}
	(0.085)	(0.011)	(0.079)	(0.008)	(0.080)	(0.028)
$\log \bar{P}_{XRP}$	0.034	0.071^{*}	-0.025	0.151^{***}	-0.146	0.045
	(0.112)	(0.037)	(0.116)	(0.031)	(0.302)	(0.054)
$\log \bar{P}_{DOGE}$	0.161^{**}	0.103^{***}	0.151^{***}	0.112^{***}	0.151^{***}	0.119^{***}
	(0.023)	(0.003)	(0.020)	(0.003)	(0.020)	(0.004)
σ_{BTC}	1.229^{***}	0.638	1.175^{***}	1.373^{***}	1.178^{**}	1.379^{***}
	(0.441)	(0.499)	(0.447)	(0.358)	(0.444)	(0.360)
σ_{ETH}	0.806^{***}	0.352	0.773^{**}	0.881^{***}	0.771^{**}	0.874^{***}
	(0.308)	(0.349)	(0.311)	(0.248)	(0.309)	(0.251)
σ_{XRP}	0.553^{***}	1.211^{***}	0.504^{***}	0.882^{***}	0.566^{***}	1.043^{***}
	(0.129)	(0.199)	(0.116)	(0.113)	(0.132)	(0.155)
σ_{DOGE}	0.346^{**}	0.971^{***}	0.310^{**}	0.613^{***}	0.363^{***}	0.750^{***}
	(0.135)	(0.195)	(0.121)	(0.111)	(0.136)	(0.147)
Const. (μ)	-	0.007	-	-0.054^{***}	-	0.057
		(0.020)		(0.019)		(.050)
C_{ECT} ($-\pi_0$)	-	0.477^{***}	-	0.707^{***}	-	0.712^{***}
		(0.099)		(0.104)		(0.099)

***, **, and * represent significance at the 1%, 5%, and 10% level, respectively.

6.3.2.1 Short-run multipliers

First, we consider the short-run multipliers ($\frac{\partial \log N_d}{\partial V_{X,d}}$), the immediate response of daily registration ($\log N_d$) to the change in an explanatory variable ($V_{X,d}$). Table 7 clearly shows that Bitcoin's average daily price increase and realized volatility have the largest immediate impact on daily user registrations. This result is consistent with the prevailing belief that Bitcoin drives cryptocurrency investments. In fact, a 1.0% increase in the daily average BTC price will cause a 1.1–1.3% increase in user registrations on the same day, and higher volatility can drive registrations further up.

On the other hand, Ripple (XRP) and Dogecoin (DOGE) show smaller immediate impacts from daily average prices and realized volatilities. DOGE appreciation shows positive correlations with user registrations in Models 3–5, but the effect magnitude is roughly one-tenth that of the BTC price. XRP's price changes do not seem to have a significant effect on user registrations.

A potential explanation for these sharp differences across cryptocurrencies lies in their respective popularity. Price swings in Bitcoin and Ethereum gain a lot more media exposure than other cryptocurrencies, which explains the much stronger correlation between the price of these currencies, and the changes in user registrations. On the other hand, although Dogecoin's social media popularity skyrocketed in early 2021, we do not observe a strong direct immediate impact on user registrations; presumably, because this popularity did not immediately percolate to more mainstream media.

6.3.2.2 Long-run multipliers

Next, we consider the long-run multipliers for each cryptocurrency ($\lim_{k\to\infty}\sum_{l=0}^{k}\frac{\partial \log N_{d+k}}{\partial V_{X,d+l}}$), the cumulative influence of the persistent change in an explanatory variable (V_X) on the daily registration ($\log N$). They show an interesting contrast to short-run multipliers.

First, we can observe, in Model 5, a reduction in the long-run multiplier for XRP's daily average price when controlling for the SEC Ripple litigation. In Models 3 and 4, where the indicator variable I_{SEC} is absent, the long-run multiplier is 0.071 (p-value = 0.053) and 0.151 (p-value $\simeq$ 0.000), indicating the influence is either significant (Model 4), or very close to being significant at the 5% level (Model 3). However, the long-run multiplier for XRP is insignificant even at the 10% level in Model 5. This result, combined with insignificant short-run multipliers, indicates that the XRP price trends lost any importance as a potential investor decision criterion, after the SEC litigation was publicly announced. That is, potential investors basically stopped considering XRP prices when thinking about whether they should join in the derivatives market. Incidentally, this litigation is still proceeding at the time of writing, and is not expected to be resolved between Q3 2023 at the earliest; whether new investors are still ignoring XRP prices in their decision-making, or whether the situation has reverted to what it was before the public announcement of the suit is an interesting open question. (v_2 in Figure 4 hints at a possible return to a state of affairs similar to that before the SEC litigation.)

Regarding realized volatilities, the long-run multipliers show that XRP and DOGE have larger values than BTC and ETH in Model 3. However, in Models 4 and 5, BTC shows the largest impact in both daily average price and realized volatilities, indicating the importance of BTC price also with respect to long-term effects. This implies that not explicitly including the effects of regulatory measures (especially the one in May) would result in a large estimate of the impact of less prominent coins.

6.3.2.3 Cumulative marginal effect

Figure 6 shows the cumulative marginal effect ($\sum_{l=0}^{k}\frac{\partial \log N_{d+k}}{\partial V_{X,d+l}}$) of the daily average prices and realized volatilities in Models 4 and 5. As we discussed in Section 5, the cumulative marginal effect can be interpreted in two ways. First, it denotes the cumulative marginal effect of the change in an explanatory variable (V_X) lasting k days. It also shows the cumulative effect of the change in an explanatory variable that happens today over the future k days (because $\sum_{l=0}^{k} \frac{\partial \log N_{d+k}}{\partial V_{d+l}} = \sum_{l=0}^{k} \frac{\partial \log N_{d+l}}{\partial V_d}$). The result for daily average prices

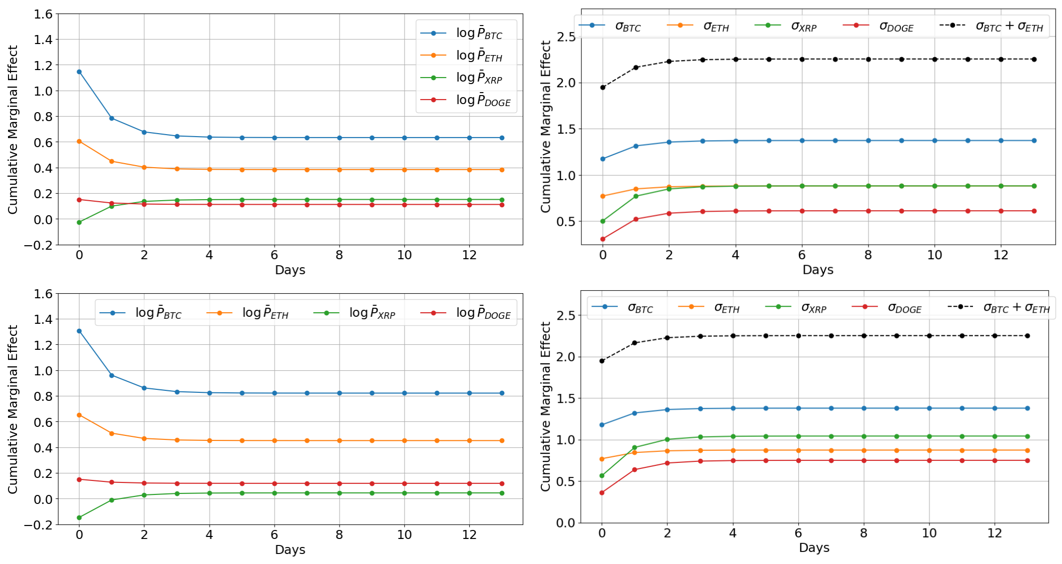


Figure 6 The cumulative marginal effect of the daily average prices and realized volatilities up to two weeks in Models 4 (top panels) and 5 (bottom panels), The black dashed line shows the sum of cumulative marginal effects for BTC and ETH.

shows that the effect of price change peaks immediately; the maximum influence comes on the day the price rises except for XRP (whose prices, as discussed above, do not have a significant short-term impact), and the cumulative effects plateau soon thereafter. In short, user registration increases by a lot immediately, and, then, the positive influence gradually decreases.

The effects of the realized volatility also peak within a few days. However, contrary to the decreasing trend in daily average prices, the cumulative effects pile up as time goes on. The cumulative effects in major coins, BTC and ETH, have a relatively slight gradient since the largest impacts manifest themselves on the same day. This means potential investors immediately react to a volatile situation. Given the chained volatility increase (*volatility clustering*) between BTC and ETH (and others) documented in several pieces of literature [24, 25] – in short, volatility of major coins foster volatile conditions for less prominent currencies as well – the sum of the influences of these coins (black dashed line in Figure 6) seemingly has a measurable market impact. In contrast, the cumulative effects of XRP and DOGE's realized volatilities accumulate by a large number on the next day and the day after that. In short, it takes a longer time for novice crypto investors to digest a volatile situation for relatively minor coins. This is an unsurprising result: contrary to high volatility in BTC and ETH prices, which can attract high publicity in both traditional media and social media, high volatility in less prominent coins, such as XRP and DOGE, will attract the attention of fewer people, which in turn will make its immediate effect more muted. For instance, as noted above, Dogecoin became a social media darling in early 2021, but it took a while for this excitement to propagate to mainstream media, and drive outside investors into cryptocurrency trading.

7 Conclusion

From ranking data on the performance of more than eight million investors in a major cryptocurrency derivatives exchange, we estimated the evolution of the number of market participants from October 1, 2020 to July 20, 2021.

We graphically observed that the daily increase in the number of users seemed to exhibit a strong correlation with major cryptocurrency prices. We formalized this result using the high descriptive capabilities of the autoregressive distributed lag (ARDL) model with Principal Component Analysis (PCA), which accounts for the idiosyncrasies of our data – numerous explanatory variables are not stationary, and are highly correlated.

We empirically analyzed the relationship between the daily user registrations and metrics related to four major cryptocurrencies, Bitcoin, Ethereum, Ripple, and Dogecoin. First, we showed evidence of a long-run equilibrium relationship between the daily registration increase and the prices of the selected cryptocurrencies. The relation is useful for estimating the number of cryptocurrency investors from publicly available price data.

Second, our analysis shows the significant influence of cryptocurrency prices on investor behavior. High price increases and volatility, in general, have the largest impact on user registration on the same day. Among the selected cryptocurrencies, the daily average price of Bitcoin is the largest contributor; this is unsurprising given Bitcoin's leading status among cryptocurrencies. Ethereum prices also significantly impact the daily user registration. In contrast, our analysis shows that Dogecoin prices have a significant but relatively small influence on user registration. A striking result of our analysis is that the impact of Ripple price fluctuations disappears when we control for the SEC litigation against Ripple Labs, Inc. Also, our regression suggests that this lawsuit, and the Chinese government's statements on tightening regulation on cryptocurrency mining and trading have a significant negative impact on user registration. These results indicate the powerful influence of regulatory measures on investor behavior.

Our regression analysis also evidences the impact of price volatility. All coins we selected show significant short-run and long-run effects of volatility on user registrations. This result is consistent with a common narrative that speculation is the primary reason for investors to start investing, so high volatility will attract more people to cryptocurrency exchanges.

However, our analysis also paints a more nuanced picture of the impact of volatility. Volatility effects considerably accumulate over time for relatively minor coins, while they are much more immediate for major cryptocurrencies. This hints at differences in information propagation speed: prominent coins are constantly scrutinized and trends are publicized in real-time, while news updates about less prominent coins initially only reach smaller circles of enthusiasts, mostly on social media, before eventually percolating to the mainstream.

As a limitation, we did not comprehensively assess regulatory measures taken in jurisdictions besides the USA and China. Investor reactions may differ depending on coin specifics, regulation relevance, and jurisdictional importance to exchanges and derivatives trading. However, while limited, our analysis clearly documents examples of the critical influence regulators can have on investor behavior.

Overall, our analysis paints a far more nuanced picture than the simplistic narrative that cryptocurrency derivatives are purely fueled by short-term speculation. Our empirical analysis instead shows potentially complex relationships between prices, volatility, and other factors such as regulatory issues. We hope this could be a starting point to help better understand investors (especially individuals) decisions to participate in cryptocurrency derivative markets, despite the odds being frequently stacked against smaller participants [44].

References

1 Hirotogu Akaike. *Information Theory and an Extension of the Maximum Likelihood Principle*, pages 199–213. Springer Series in Statistics. Springer, 1998.

2 Carol Alexander, Jaehyuk Choi, Hamish R. A. Massie, and Sungbin Sohn. Price discovery and microstructure in ether spot and derivative markets. *International Review of Financial Analysis*, 71:101506, 2020. `doi:10.1016/j.irfa.2020.101506`.

3 Carol Alexander, Jaehyuk Choi, Heungju Park, and Sungbin Sohn. Bitmex bitcoin derivatives: Price discovery, informational efficiency, and hedging effectiveness. *Journal of Futures Markets*, 40(1):23–43, 2020. `doi:10.1002/fut.22050`.

4 Donald W. K. Andrews. Laws of large numbers for dependent non-identically distributed random variables. *Econometric Theory*, 4(3):458–467, 1988. `doi:10.1017/S0266466600013396`.

5 Caitlin Ostroff Areddy and James T. Bitcoin, ether prices continue falling after china spurs regulatory fears. *Wall Street Journal*, May 2021. URL: `https://www.wsj.com/articles/bitcoin-ether-prices-continue-falling-on-regulatory-fears-11621611655`.

6 Dirk G. Baur, KiHoon Hong, and Adrian D. Lee. Bitcoin: Medium of exchange or speculative assets? *Journal of International Financial Markets, Institutions and Money*, 54:177–189, 2018. `doi:10.1016/j.intfin.2017.12.004`.

7 BRUNO BIAIS, CHRISTOPHE BISIÈRE, MATTHIEU BOUVARD, CATHERINE CASAMATTA, and ALBERT J. MENKVELD. Equilibrium bitcoin pricing. *The Journal of Finance*, 78(2):967–1014, 2023. `doi:10.1111/jofi.13206`.

8 Nicola Borri. Conditional tail-risk in cryptocurrency markets. *Journal of Empirical Finance*, 50:1–19, 2019. `doi:10.1016/j.jempfin.2018.11.002`.

9 George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

10 Brad Chase and Ethan MacBrough. Analysis of the xrp ledger consensus protocol, 2018. `doi:10.48550/ARXIV.1802.07242`.

11 Nicolas Christin. Traveling the silk road: a measurement analysis of a large anonymous online marketplace. In *Proceedings of the 22nd international conference on World Wide Web*, WWW '13, pages 213–224, New York, NY, USA, 2013. Association for Computing Machinery. `doi:10.1145/2488388.2488408`.

12 Pavel Ciaian, Miroslava Rajcaniova, and d'Artis Kancs. The economics of bitcoin price formation. *Applied Economics*, 48(19):1799–1815, 2016. `doi:10.1080/00036846.2015.1109038`.

13 Lin William Cong, Ye Li, and Neng Wang. Tokenomics: Dynamic adoption and valuation. *The Review of Financial Studies*, 34(3):1105–1155, 2021. `doi:10.1093/rfs/hhaa089`.

14 David A. Dickey and Wayne A. Fuller. Likelihood ratio statistics for autoregressive time series with a unit root. *Econometrica*, 49(4):1057–1072, 1981. `doi:10.2307/1912517`.

15 Paul Kiernan Duehren and Andrew. Bitcoin price surges on biden's crypto executive order. *Wall Street Journal*, 2022. URL: `https://www.wsj.com/articles/biden-to-order-study-of-cryptocurrency-risk-creation-of-u-s-digital-currency-11646823600`.

16 Financial Stability Board. Assessment of risks to financial stability from crypto-assets, 2022. URL: `https://www.fsb.org/2022/02/assessment-of-risks-to-financial-stability-from-crypto-assets/`.

17 Financial Stability Board. Fsb work programme for 2022, 2022. URL: `https://www.fsb.org/2022/03/fsb-work-programme-for-2022/`.

18 Maddala G.S. *Introduction to Econometrics, 3rd Edition.* John Wiley & Sons, 2007.

19 James Hamilton. *Time Series Analysis.* Princeton University Press, Princeton, NJ, 1994.

20 Bruce Hansen. *Econometrics.* Princeton University Press, Princeton, NJ, 2022.

21 Anna Hirtenstein. Binance crypto exchange ordered to cease u.k. activities. *Wall Street Journal*, June 2021. URL: `https://www.wsj.com/articles/binance-crypto-exchange-ordered-to-cease-u-k-activities-11624812672`.

22 Tara Iyer. Cryptic connections. Technical report, International Monetary Fund, 2022. URL: `https://www.imf.org/en/Publications/global-financial-stability-notes/Issues/2022/01/10/Cryptic-Connections-511776`.

23 Ryan James. Xrp continues gains following 40% gain on saturday, April 2021. URL: `https://beincrypto.com/xrp-continues-gains-following-40-gain-on-saturday/`.

24 Paraskevi Katsiampa, Shaen Corbet, and Brian Lucey. High frequency volatility co-movements in cryptocurrency markets. *Journal of International Financial Markets, Institutions and Money*, 62:35–52, September 2019. `doi:10.1016/j.intfin.2019.05.003`.

25 Paraskevi Katsiampa, Shaen Corbet, and Brian Lucey. Volatility spillover effects in leading cryptocurrencies: A bekk-mgarch analysis. *Finance Research Letters*, 29:68–74, June 2019. `doi:10.1016/j.frl.2019.03.009`.

26 Daisuke Kawai, Alejandro Cuevas, Bryan Routledge, Kyle Soska, Ariel Zetlin-Jones, and Nicolas Christin. Is your digital neighbor a reliable investment advisor? In *Proceedings of the ACM Web Conference 2023*, WWW '23, pages 3581–3591, New York, NY, USA, 2023. Association for Computing Machinery. `doi:10.1145/3543507.3583502`.

27 Ladislav Kristoufek. Bitcoin meets google trends and wikipedia: Quantifying the relationship between phenomena of the internet era. *Scientific Reports*, 3(1):3415, 2013. `doi:10.1038/srep03415`.

28 Yukun Liu and Aleh Tsyvinski. Risks and returns of cryptocurrency. *The Review of Financial Studies*, 34(6):2689–2727, 2021. `doi:10.1093/rfs/hhaa113`.

29 James Mackintosh. Behind bitcoin price gyrations: Rational action and wild speculation. *Wall Street Journal*, May 2021. URL: `https://www.wsj.com/articles/bitcoin-is-the-apogee-of-rational-speculation-11621524833`.

30 D. L. McLeish. Dependent Central Limit Theorems and Invariance Principles. *The Annals of Probability*, 2(4):620–628, 1974. `doi:10.1214/aop/1176996608`.

31 Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 127–140, New York, NY, USA, 2013. Association for Computing Machinery. `doi:10.1145/2504730.2504747`.

32 T. Moore and N. Christin. Beware the middleman: Empirical analysis of Bitcoin-exchange risk. In *Proceedings of IFCA Financial Cryptography'13*, Okinawa, Japan, April 2013.

33 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: `https://bitcoin.org/bitcoin.pdf`.

34 Emiliano Pagnotta and Andrea Buraschi. *An Equilibrium Valuation of Bitcoin and Decentralized Network Assets*. SSRN Scholarly Paper, Rochester, NY, 2018. URL: `https://papers.ssrn.com/abstract=3142022`.

35 Emiliano S Pagnotta. Decentralizing money: Bitcoin prices and blockchain security. *The Review of Financial Studies*, 35(2):866–907, January 2021. `doi:10.1093/rfs/hhaa149`.

36 M. Hashem Pesaran, Yongcheol Shin, and Richard J. Smith. Bounds testing approaches to the analysis of level relationships. *Journal of Applied Econometrics*, 16(3):289–326, 2001. `doi:10.1002/jae.616`.

37 B. Pfaff. *Analysis of Integrated and Cointegrated Time Series with R*. Springer, New York, second edition, 2008. ISBN 0-387-27960-1. URL: `https://www.pfaffikus.de`.

38 Peter C. B. Phillips and Pierre Perron. Testing for a unit root in time series regression. *Biometrika*, 75(2):335–346, 1988. `doi:10.2307/2336182`.

39 Thomas P. Ryan. *Modern Regression Methods*, chapter 4, pages 146–189. John Wiley & Sons, Ltd, 2008. `doi:10.1002/9780470382806.ch4`.

40 Linda Schilling and Harald Uhlig. Some simple bitcoin economics. *Journal of Monetary Economics*, 106:16–26, 2019. `doi:10.1016/j.jmoneco.2019.07.002`.

41 Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. URL: `https://www.jstor.org/stable/2958889`.

42 Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

43 Michael Sockin and Wei Xiong. A model of cryptocurrencies. Working Paper 26816, National Bureau of Economic Research, 2020. doi:10.3386/w26816.

44 K. Soska, J.-D. Dong, A. Khodaverdian, A. Zetlin-Jones, B. Routledge, and N. Christin. Towards understanding cryptocurrency derivatives: A case study of BitMEX. In *Proceedings of the 30th Web Conference (WWW'21)*, 2021.

45 Andrew Urquhart. What causes the attention of bitcoin? *Economics Letters*, 166:40–44, 2018. doi:10.1016/j.econlet.2018.02.017.

46 U.S. Commodity and Futures Trading Commission. Cftc charges binance and its founder, changpeng zhao, with willful evasion of federal law and operating an illegal digital asset derivatives exchange | cftc, March 2023. URL: From:https://www.cftc.gov/PressRoom/PressReleases/8680-23.

47 U.S. Commodity Futures Trading Commission. Cftc charges sam bankman-fried, ftx trading and alameda with fraud and material misrepresentations, August 2022. URL: https://www.cftc.gov/PressRoom/PressReleases/8638-22.

48 U.S. Commodity Futures Trading Commission. Federal court orders bitmex to pay $100 million for illegally operating a cryptocurrency trading platform and anti-money laundering violations, August 2022. URL: https://www.cftc.gov/PressRoom/PressReleases/8412-21.

49 U.S. Securities and Exchange Commission. Remarks before the aspen security forum, 2021. URL: https://www.sec.gov/news/public-statement/gensler-aspen-security-forum-2021-08-03.

50 U.S. Securities and Exchange Commission. Sec charges ripple and two executives with conducting $1.3 billion unregistered securities offering, 2021. URL: https://www.sec.gov/news/press-release/2020-338.

51 U.S. Securities and Exchange Commission. Sec files 13 charges against binance entities and founder changpeng zhao, June 2023. URL: https://www.sec.gov/news/press-release/2023-101.

52 William W.S. Wei. *Time Series Analysis*. Oxford University Press, 2013. doi:10.1093/oxfordhb/9780199934898.013.0022.

A Cryptocurrency prices

This section shows the daily average prices and realized volatilities of the cryptocurrencies we consider: Bitcoin (BTC), Ethereum (ETH), Ripple (XRP), and Dogecoin(DOGE).

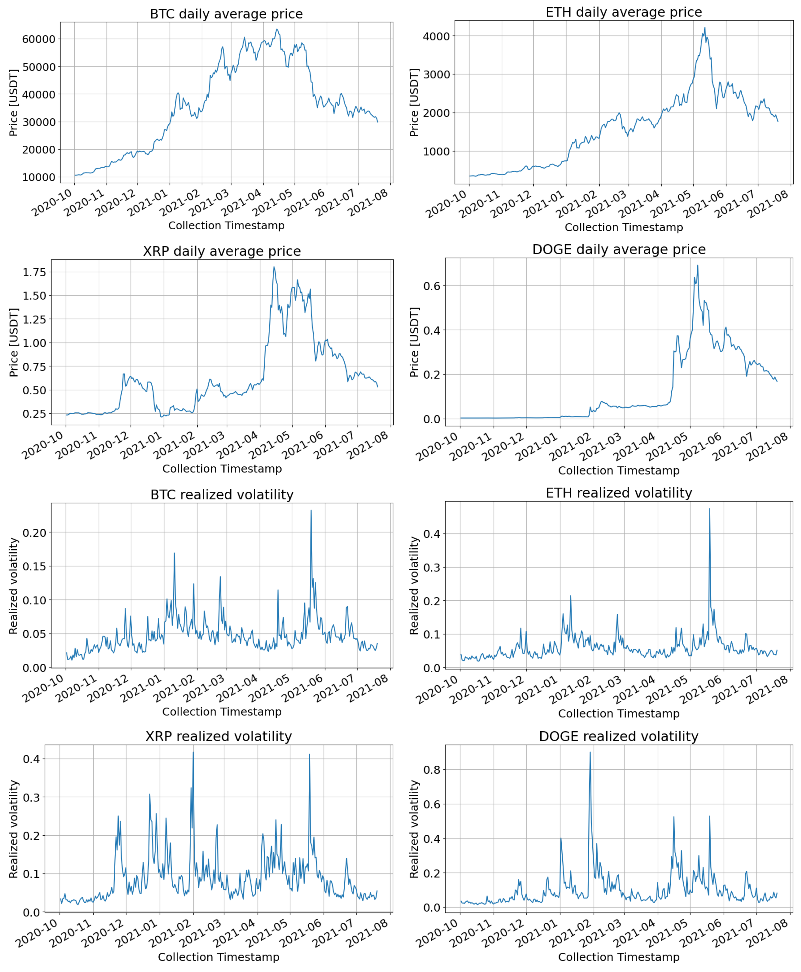


Figure 7 The daily average prices (upper panels) and realized volatilities (lower panels).

B Convergence of marginal effects

This section considers the convergence of marginal effects $\left(\lim_{k\to\infty} \frac{\partial \widehat{\log N_{d+k}}}{\partial V_{X,d}} = 0\right)$.

We can derive the difference equation for a marginal effect from Eqn. (1) and substitute the coefficients with the estimates in Models 1–5 summarized in Table 6. For example, the equation for the first principal component of daily average prices ($v_{1,d}$) in Model 5 is:

$$\frac{\partial \widehat{\log N_{d+k}}}{\partial v_{1,d}} = (1+\pi_0)\frac{\partial \widehat{\log N_{d+k-1}}}{\partial v_{1,d}} = (1-0.712)\frac{\partial \widehat{\log N_{d+k-1}}}{\partial v_{1,d}} \quad (k \geq 2)\,. \tag{5}$$

It clearly shows that the marginal effect converges to zero as $k \to \infty$. We can similarly consider the convergence of every marginal effect and confirm that all marginal effects converge to zero in the limit $k \to \infty$. This means we can consider long-run multipliers for all explanatory variables.

DeFi Lending During The Merge

Lioba Heimbach
ETH Zürich, Switzerland

Eric Schertenleib
ETH Zürich, Switzerland

Roger Wattenhofer
ETH Zürich, Switzerland

Abstract

Lending protocols in decentralized finance enable the permissionless exchange of capital from lenders to borrowers without relying on a trusted third party for clearing or market-making. Interest rates are set by the supply and demand of capital according to a pre-defined function. In the lead-up to *The Merge*: Ethereum blockchain's transition from *proof-of-work (PoW)* to *proof-of-stake (PoS)*, a fraction of the Ethereum ecosystem announced plans of continuing with a PoW-chain. Owners of ETH – whether their ETH was borrowed or not – would hold the native tokens on each chain. This development alarmed lending protocols. They feared spiking ETH borrowing rates would lead to mass liquidations which could undermine their viability. Thus, the decentralized autonomous organization running the protocols saw no alternative to intervention – restricting users' ability to borrow.

We investigate the effects of the merge and the aforementioned intervention on the two biggest lending protocols on Ethereum: AAVE and Compound. Our analysis finds that borrowing rates were extremely volatile, jumping by two orders of magnitude, and borrowing at times reached 100% of the available funds. Despite this, no spike in mass liquidations or irretrievable loans materialized. Further, we are the first to quantify and analyze *hard-fork-arbitrage*, profiting from holding debt in the native blockchain token during a hard fork. We find that arbitrageurs transferred tokens to centralized exchanges which at the time were worth more than 13 Mio US$, money that was effectively extracted from the platforms' lenders.

2012 ACM Subject Classification General and reference → Measurement; General and reference → Empirical studies; Applied computing → Economics

Keywords and phrases blockchain, Ethereum, lending protocol, hard fork

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.9

1 Introduction

Participants in financial markets borrow or lend for a plethora of reasons: raising capital for investments such as buying a house, saving for retirement, or short-selling securities. Particularly consequential for the stability of the financial market is leveraged trading. Thereby, a trader takes on leverage by borrowing from a counterparty, typically a broker or bank, to buy financial securities. The lender must approve the borrowing and puts certain restrictions in place to safeguard their funds (margin requirements). Given the counterparty risks, i.e., the risk that the other party defaults on their contractual obligation, regulators closely monitor these activities and enforce certain restrictions to reduce the likelihood of major financial upheaval.

Decentralized Finance (DeFi) promises to offer financial services to users without requiring prior clearance or a known and trustworthy counterparty. Instead, DeFi is built using *smart contracts*, i.e., executable code on the blockchain. Lending protocols have a central role in the DeFi protocol space. They allow anyone to become a lender by depositing their funds in the protocol. Further, anyone can borrow funds as long as their deposited assets exceed their borrowed funds in value by a pre-defined margin (over-collateralization). The restriction to over-collateralize loans on lending protocols is their key to unlocking trustless loans. Additionally, the over-collateralization margin aims to ensure that there is enough time for a position at risk of becoming under-collateralized to become liquidated in time for the debt to be recovered.

Thus, it is crucial for lending protocols to correctly assess the risks of the crypto assets they allow as collateral in the market and to set the margin accordingly. Lending protocols strive to find the right balance between (i) offering competitive rates to borrowers and (ii) low risks for lenders. As the risks associated with the various collateral assets are likely to change over time, lending protocols can adjust various risk-related parameters, such as the over-collateralization margin. These changes are generally discussed and decided by the *decentralized autonomous organization (DAO)* that governs the protocol.

The Ethereum blockchain is the birthplace of DeFi and the home of the leading lending protocols in terms of total value locked [27]. Initially, the consensus mechanism of the Ethereum blockchain relied on the energy-intensive *proof-of-work (PoW)*. Ethereum had planned for years to switch to the more energy-efficient *proof-of-stake (PoS)*. *The Merge*, which was executed in September 2022, marked the end of PoW and the start of PoS. However, in the lead-up to the merge, there was opposition from parts of the Ethereum mining community whose business model relied on PoW. They pushed for a hard fork that would retain a PoW blockchain. Thus, even though the vast majority of the Ethereum community announced that they would switch to PoS, it was unclear how the value of ETH would be distributed between the two blockchains.

The approaching merge and possibility of a hard fork, as well as the resulting distribution of value between the two blockchains posed a challenge to lending protocols. Anyone who held ETH on the blockchain in the last block before the merge would then own the same number of tokens on both the PoS chain and the PoW chain after the merge. Thus, given that future markets indicated that the PoW token (ETHW) would retain a value of a few percentage points of its PoS counterpart, some users borrowed ETH ahead of the merge in order to receive both tokens after the merge. We will refer to this as *hard-fork-arbitrage*, a form of event-driven arbitrage.

Thus, protocols expected the demand for ETH borrowing to increase drastically and, as a result, expected both ETH borrowing rates and the utilization, the ratio between the ETH loans and ETH liquidity, of the ETH market to skyrocket. Furthermore, given the intertwined nature of DeFi protocols and the central position lending protocols occupy therein, the stability of these platforms is essential.

In response, the DAOs governing AAVE and Compound, the two largest lending protocols on the Ethereum blockchain, decided to intervene. While AAVE paused ETH borrowing entirely ahead of the merge, Compound adjusted the risk parameters that determine the borrowing rate and capped ETH borrowing. This intervention, however, led to market distortions that adversely affected lenders, who were effectively on the losing side of the hard-fork-arbitrage. Furthermore, the lending market dried up as protocols ceased to have any available liquidity.

Our Contributions

We analyze lending protocols during a critical time period. The execution of the merge on the Ethereum blockchain tested the resiliency of lending protocols toward significant external events. Thus, it provides a crucial case study of challenges faced by DeFi protocols under extraordinary circumstances. We analyze the effects of the merge on lending protocols as well as their attempts to mitigate borrowing rate spikes. They feared that such rate spikes would cause both mass liquidations, i.e., enforced repayment of the debt, and the accumulation of bad debt, i.e., debt that is not over-collateralized, leading to losses for lenders and potentially harming their own viability. Our analysis focuses on AAVE and Compound, the biggest DeFi lending protocols. Together AAVE and Compound account for more than 85% of the volume locked on lending protocols on the Ethereum blockchain, the home of most DeFi applications [27].

- We find that the changes the protocols implemented, capping borrowing, may have helped prevent mass liquidations and the accumulation of bad debt but were only partially successful in keeping rates at normal levels.
- We show that the protocols failed to adequately compensate their lenders, thus placing them at the losing end. The beneficiaries were arbitrageurs, including the now infamous Alameda research, who extracted in excess of 300,535 ETHW tokens in what we term hard-fork-arbitrage. To the best of our knowledge, we are the first to study and quantify hard-fork-arbitrage.
- Finally, we find that the widespread use of ETH staked through LIDO as collateral posed a grave danger to the entire Ethereum blockchain as staking power could have been gobbled up at a significant discount.

2 Related Work

DeFi lending emerged in 2017 and first became popularized during the 2020 DeFi summer. Bartoletti et al. [3] provide a systematization of existing knowledge regarding DeFi lending protocols. Further, they offer a formal framework to model the interactions between users in lending pools. In the first empirical study of lending protocols, Gudgeon et al. [22] study different interest rate rules across DeFi lending protocols. Their work analyzes the historical responses of the markets to their liquidity depths. We, on the other hand, investigate the response of lending protocols to the merge and illustrate their reliance on intervention.

A recent line of work studies the risks stemming from the increasing complexity of DeFi protocol compositions. Tolmach et al. [41] provide a formal analysis of DeFi composability and propose a technique for efficient property verification. A measurement study by Kitzler et al. [23] empirically analyzes and visualizes DeFi compositions – demonstrating the intertwined nature of DeFi protocols. Wachter et al. [46] measure growing asset composability as a proxy for the interoperability of the DeFi applications. This interoperability poses a systemic risk to the DeFi ecosystem, given its resulting convolution.

The central position of lending in DeFi and the aforementioned intertwined nature of DeFi can lead to increased sensitivity to shocks in the ecosystem. Chiu et al. [5] outline that DeFi lending protocols make cryptocurrency prices more sensitive to fundamental shocks. Their work finds that intervention to provide risk management in DeFi lending protocols may improve efficiency and stability while compromising decentralization. In our work, we analyze how actions and interventions taken by DAOs panned out and affected market participants.

DeFi lending is generally used to facilitate crypto asset price speculation as opposed to real economy lending, as highlighted in a recent bulletin by Aramonte et al. [2]. As the authors and others [48, 9] point out, the borrowed funds from lending can be reused as collateral to take out additional loans leading to increased leverage. Such leverage spirals, which are in part made possible by DeFi composability, exasperate the vulnerability of lending protocols towards external events. In contrast to these works, we study the response to and resilience of lending protocols toward external events of lending protocols.

An active line of research documents and measures multiple attacks and arbitrage opportunities exploiting the design of lending protocols [11, 37, 50]. A particular focus is placed on liquidations. While Perez et al. [33] study the efficiency of liquidations, Qin et al. [36] study optimal liquidation strategies. Our work studies and quantifies a novel and previously unstudied form of arbitrage on lending protocols, which we term hard-fork-arbitrage.

3 Background

In the following, we discuss the specifics regarding the merge on the Ethereum blockchain (cf. Section 3.1), the PoW hard fork (cf. Section 3.2), as well as the mechanisms of lending protocols (cf. Section 3.3) and DAOs (cf. Section 3.4).

3.1 The Merge

On 15 September 2022, the merge [13] was executed on the Ethereum mainnet. The merge marked the end of energy-intensive PoW and the start of energy-efficient PoS for the Ethereum blockchain.

PoW, originally proposed by Nakamoto [32], is the most established consensus for blockchains. Miners must solve a computationally intensive puzzle, and the winner of the puzzle updates the blockchain by appending the newest block. Due to the energy-intensive nature of PoW, there have long been calls to reduce the energy consumption of blockchains.

The most established blockchain consensus alternative to PoW is the energy-efficient PoS, which was adopted by Ethereum during the merge. Stakers, the miner counterpart for PoS blockchains, offer their funds as collateral for the chance to be selected as a block's validator. For every block, a staker is selected as the block's validator. The chance of being selected as a validator in each round is proportional to their locked-up funds.

3.2 EthereumPoW

Unsurprisingly, there was uproar from the Ethereum mining community before the merge. The transition rid the miners of their revenue stream and thereby forced them to scrap their hardware or move to other PoW chains. Thus, there were multiple efforts to fork the Ethereum blockchain and create a spinoff, PoW version [16, 14]. Chandler Guo, a prominent cryptocurrency miner, led the most notable effort to the Ethereum blockchain. The resulting chain is known as EthereumPoW (ETHW) [16].

Thus, ahead of the merge, a chain split into ETH and ETHW was anticipated. Anyone holding ETH on the original chain right before the merge would automatically receive an equal amount of ETHW tokens after the fork. Speculations surrounding the upcoming chain split led to a trading start of ETHW ahead of the merge. ETHW started trading on 9 August 2022 around 97 US$ and was trading around 45 US$ during the merge.

There have been more than twelve Ethereum hard-forks in the past [40], the most prominent blockchain that resulted from earlier forks being Ethereum Classic (ETC) [12]. This specific fork was anticipated to be particularly challenging for EthereumPoW. Both the rise of DeFi and the prevalence of asset-backed tokens[1] on the Ethereum chain complicated matters. For example, the organizations behind USDT and USDC, the biggest stablecoins

[1] Asset-backed tokens are tokens that derive their value from underlying assets that are not necessarily on the same blockchain. E.g., the stablecoin USDC is a token emitted by an organization that promises that for each USDC they hold one US$.

in terms of market capitalization [6], announced that they would support the transition to PoS [39, 38]. As they are both asset-backed, users would hold equal amounts of their USDC/USDT tokens on both chains right after the merge, but the tokens on the ETHW chain would be worthless as they are no longer backed. Note that not only stablecoins but, in fact, the vast majority of ERC20 tokens are asset-backed and were therefore expected to become worthless on the ETHW chain.

3.3 Lending Protocols

Lending protocols are among the most successful DeFi applications. They facilitate trustless and decentralized cryptocurrency loans. In traditional finance, one generally receives loans from financial institutions such as banks. As security, banks require collateral for the loan, for example, the house in the case of a mortgage. Additionally, the conditions of the loan are negotiated and vary across loans. DeFi, on the other hand, automates lending by relying on smart contracts. Lending protocols allow anyone to become a lender. By locking their cryptocurrency assets in the protocol's smart contract users become *liquidity providers*. In exchange for providing capital, they earn interest on their assets. Thus, liquidity providers are the lynchpin of lending protocols as they provide the capital.

Interest payments are made by borrowers who take out loans against their locked cryptocurrency collateral. More specifically, borrowers can take out loans without prior clearance by depositing collateral, as long as the collateral is greater in value, i.e., the loans are over-collateralized. Over-collateralization, thus, is the key behind the trustless nature of lending protocols, as it protects lenders against downside price movements of the borrowers' collateral. Once a loan is no longer sufficiently over-collateralized the borrower is incentivized to adjust the position. Generally, loans are insufficiently collateralized if the value of the collateral does not exceed the debt value by more than 20%. This margin does vary depending on the protocol's risk assessment of the collateral. In case the borrower does not react, the position will likely be closed by *liquidators* at a cost for the borrower.

Note that loans on lending protocols are generally for an indefinite time, as interest payments are made periodically. Additionally, the interest payments are generally variable and typically dependent on the asset borrowed as well as the utilization. The utilization at time t of an asset is given as $U_t = L_t/D_t$, where L_t denotes the total outstanding loans and D_t denotes the deposits. We will go through the specifics for AAVE and Compound in the following. AAVE V2 and Compound V2 were the largest and newest markets for ETH[2] borrowing around the time of the merge. Together they currently account for more than 85% of the volume locked in lending protocols [27].

Borrowers on AAVE can choose between stable interest payments and variable interest rate payments. Both are charged periodically, at every time step, and depend on the asset's utilization. Note that the interest rate is charged periodically by simply adjusting the balance of the debt tokens held by the borrowers (debt is compounded). The interest rate a borrower is charged at time t is given as follows

[2] Users technically borrow wrapped ETH (WETH), an ERC20 compatible version of ETH, on the two protocols. As WETH is simply a wrapped version of ETH that has virtually the same value as ETH, we refer to WETH as ETH throughout.

$$r_t = \begin{cases} r_0 + \dfrac{U_t}{U_{\text{optimal}}} r_{\text{slope}_1} & \text{if } U_t \leq U_{\text{optimal}}, \\ r_0 + r_{\text{slope}_1} + \dfrac{U_t - U_{\text{optimal}}}{1 - U_{\text{optimal}}} r_{\text{slope}_2} & \text{if } U_t > U_{\text{optimal}}, \end{cases}$$

where U_t is the current utilization of the asset, and r_0, r_{slope_1}, r_{slope_2}, U_{optimal} are configuration parameters. U_{optimal} is the target utilization of the protocol, once the utilization rises beyond U_{optimal} borrowing rates rise sharply. Note that both the stable and variable interest rates are computed as indicated above, but the configuration parameters for the same asset differ. The configuration parameters for ETH were set as indicated in Table 1a during the merge. Further, we draw both the stable and variable interest rates as a function of the utilization in Appendix C (cf. Figure 15). Once the utilization surpasses U_{optimal}, the interest rises at a significantly higher rate, i.e., there is a *kink* in the interest rate curve at U_{optimal}. A loan that is taken out with a variable interest rate is charged periodically according to the current variable interest rate. On the other hand, a loan that is taken out with a stable interest rate at time t, r_t^s, continues to be charged this rate. Note that the stable rate is not guaranteed to remain stable for an indefinite time period. Instead, it can be adjusted if the loan's stable rate is lower than the current supply rate received by lenders [20].

Table 1 Parameters for ETH on AAVE V2 and Compound ahead of the merge. Note that the Compound parameters were adjusted on 10 September 2022 (cf. Section 5.1).

(a) AAVE.

	U_{optimal}	r_0	r_{slope_1}	r_{slope_2}	R
stable rate	70%	3%	4%	100%	10%
variable rate	70%	0%	3%	100%	10%

(b) Compound V2.

	r_0	r_{slope_1}	R
variable rate	2%	10%	20%

Lenders, on the other hand, deposit their assets and in return receive continuous interest rate payments. Precisely, the supply rate, that is the rate they receive, at time t is given as

$$s_t = U_t(D_t^s \tilde{r}_t^s + D_t^v r_t^v)(1 - R),$$

where D_t^s is the share of stable loans, r_t^s is the average stable interest rate, D_t^v is the share of variable loans, and r_t^v the variable interest rate. Further, R is the reserve factor, which signifies the minimum proportion of borrow rate payments that flow into the protocol's treasury. Thus, the supply rate is always lower than the borrowing rate, especially when utilization is low. The difference between the two rates is the revenue source of the protocol. We note that lenders can withdraw their assets at all times, as long as the utilization allows for it, i.e., there are sufficient funds that are currently not being borrowed.

As opposed to AAVE, Compound only offers variable interest rate loans. Furthermore, while AAVE indicates annualized rates and then charges for the time the money was borrowed, Compound charges a per-block rate. However, the Compound smart contract is configured with yearly rates and assumes 2,102,400 blocks per year [8]. Throughout this work, whenever we display annualized rates for Compound, for better comparability with AAVE, we will assume 6,245 blocks per day (2,279,425 blocks per year), the average number of daily blocks ahead of the merge.

For ETH, the interest rate at time t is given as follows

$$r_t = r_0 + U_t \cdot r_{\text{slope}},$$

where U_t is again the utilization, and r_0, r_{slope} are configuration parameters[3]. We provide the parameters ahead of the merge in Table 1b. Notice that while the interest rate on Compound is higher than on AAVE for low utilization, it is significantly lower for high utilization. Similarly to AAVE, lenders on Compound deposit their assets and receive interest payments continuously. The supply rate is given as

$$s_t = r_t \cdot U_t(1 - R),$$

where r_t is the borrowing interest rate and the R is again a reserve factor.

On both AAVE and Compound, loans that are close to no longer being sufficiently over-collateralized become available for liquidation. To be precise, if the *health factor* of a position drops below 1, a position can be liquidated. A position's health factor is given as

$$H = \frac{\sum\limits_{i \in A} (C_i \cdot l_i)}{\sum\limits_{i \in A} D_i},$$

where A is the set of available assets on the platform. C_i is the position's collateral in asset i and D_i is the position's debt in asset i. Finally, l_i is the liquidation threshold for asset i, which is a configuration parameter. A position with a liquidation threshold of 75% is considered under-collateralized if the value of the debt rises above 75% in comparison to the collateral. Once a position becomes available for liquidation, its collateral is auctioned off at a discount if the liquidator repays the debt in return.

3.4 DAO

Many DeFi protocols, including AAVE and Compound, are governed by a DAO. A DAO is generally composed of the protocol's token holders, who come together to make decisions regarding the protocol according to specified rules that are enforced by the smart contract. Generally, DAOs can make changes to the protocol itself and make decisions regarding the protocol's funds.

The AAVE DAO is composed of AAVE (AAVE's native token) holders, while the Compound DAO is composed of COMP (Compound's native token) holders. Both DAOs have the power to change the lending protocol's risk parameters in order to be able to respond to changing risks regarding the market's assets. The community generally first discusses proposed changes and then decides by voting. Depending on the outcome of the vote, the changes will automatically be adopted by the protocol's smart contracts.

Finally, despite their name, DAOs are only as decentralized as the distribution of governance tokens among the actively participating users. However, an analysis of the voting power of various DAOs has shown that the voting power is generally, in effect, very concentrated [21].

[3] Note that ahead of the merge, ETH adopted Compound's standard interest rate model, i.e., the interest rate increases linearly with the utilization and does not exhibit a change in slope. This, however, was adapted in anticipation of the merge (cf. Section 5).

4 Data

We concentrate the data analysis between 9 August, the day the price for ETHW became available, and 15 October 2022, a month after the merge. With this time frame, we can observe behaviors on lending protocols and their implications. Note that we occasionally include data for shorter or longer time periods to understand general trends better or to zoom in on details. In the following, we provide a concise description of our data collection.

4.1 Ethereum

To collect data from the Ethereum blockchain, we run an Erigon [25] Ethereum archive node, i.e., a node that builds an archive of historical state. In particular, we collect data from AAVE [1] and Compound [7], the two biggest lending protocols on the Ethereum blockchain that have an ETH borrowing market. To obtain the relevant data from the lending protocols, we filter for event logs emitted by the two regarding the relevant underlying assets. We also query the historical state of the lending markets by calling the implemented functions daily through the web3.eth API [47].

Further, we follow the ETH debt borrowed on the two protocols to identify whether the debt was transferred to cryptocurrency exchanges and, thereby, likely sold. We filter through the transaction traces stored on our Erigon archive node. We utilize the Etherscan (Ethereum block explorer) Label Word Cloud [24] to obtain wallet labels and later be able to identify transfers to exchanges.

4.2 EthereumPoW

We run a full geth [17] EthereumPoW node to collect EthereumPoW blockchain data. To the best of our knowledge, geth is the only node implementation specifically for EthereumPoW.

Further, to follow the ETH debt borrowed on the two protocols after the merge, we filter through the transactions stored on our EthereumPoW node. Note that geth does not implement a trace filter. Thus, we filter the transactions and identify ETHW transfers done through a regular transaction, i.e., a transfer from one account to another. We might miss additional transfers in transactions that execute a contract but still obtain a lower bound for the ETHW transferred. We utilize the Etherscan Label Word Cloud [24] and OKLINK [19] (EthereumPoW block explorer) to obtain wallet labels. Note that the addresses owned by exchanges before the merge still belong to those exchanges on the EthereumPoW blockchain.

4.3 Price Data

We gather hourly price data for the relevant cryptocurrencies from Yahoo Finance [49] by interacting with their Python API [51]. We use Yahoo Finance price data instead of the Chainlink price oracle, as Yahoo Finance tracked the ETHW price. However, when discussing a position's health, we will utilize prices from the respective Chainlink price oracle [4].

5 Merge Anticipation

We commence the analysis by considering the ETHW price leading up to and post-merge (cf. Figure 1). Notice that ETHW's price measured in terms of US$ (in green) and ETH (in red) moves very similarly. Thus, we infer that the price movements of ETHW were considerably more pronounced than those of ETH. We also see that the price of ETHW generally falls in relation to that of ETH, with one notable exception: right before the merge, ETHW's

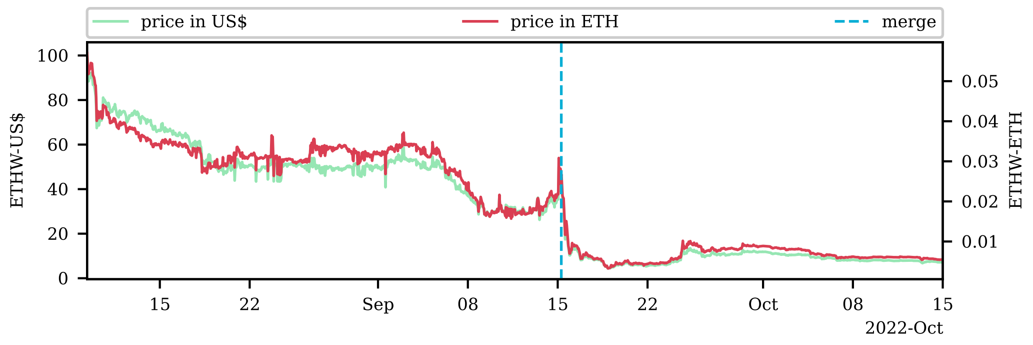


Figure 1 ETHW price in anticipation of and after the merge. The execution of the merge is marked by the dashed blue line. Notice the sharp price increase right before the merge and the dramatic price drop by more than 75% right after the merge. The line plots are quite similar as the ETH-USD price was significantly less volatile around the merge.

price measured in ETH spiked. Further, we notice that the value of ETHW was around 3% that of ETH at the time the merge was executed. Thus, everyone that held ETH right before the merge – irrespective of whether the ETH was borrowed or not – received ETHW tokens worth 3% of their holdings. In particular, one only needed to hold the ETH tokens in one's wallet for a single block, the last block prior to the merge, in order to receive an additional 3% in value. Thus, in anticipation of this event, lending protocols feared that users would take out ETH loans right before the merge in order to profit from the fork. Note that liquidity providers, with their ETH locked, did not profit from the hard fork and, thus, had the incentive to pull out their funds, further driving up utilization. As excessive borrowing activities can cause major distress to lending protocols, they intervened in order to disincentive/disallow such behavior.

5.1 Compound

On 16 August 2022, one month before the merge, the Compound community started discussing the consequences of the merge for their protocol. They devised plans to get ahead of the anticipated event [30]. The Compound community raised concerns regarding the liquidity risk for ETH ahead of the merge. In particular, they feared that DeFi users would withdraw ETH from Compound and/or borrow any available ETH. To address this, it was suggested to update the risk parameter and cap the amount of ETH that can be borrowed. After two weeks of discussion, a vote was held by the COMP token holders on 8 September 2022 [31]. The community voted in favor of the changes, and the alterations were executed two days later, on 10 September 2022. Thus, the process took a total of three weeks.

As part of the change, Compound switched the interest rate model for ETH to what they call the jump interest rate model. In this model the interest rate at time t contains a kink and is given by

$$r_t = \begin{cases} r_0 + U_t \cdot r_{\text{slope}_1} & \text{if } U_t \le U_{\text{optimal}}, \\ r_0 + U_t \cdot r_{\text{slope}_1} + (U_t - U_{\text{optimal}}) \cdot r_{\text{slope}_2} & \text{if } U_t > U_{\text{optimal}}. \end{cases}$$

Here, U_t is again the asset's current utilization, and r_0, r_{slope_1}, r_{slope_2}, U_{optimal} are configuration parameters (cf. Table 2).

Table 2 Parameters for ETH Compound after the adoption of Proposal 122 [30].

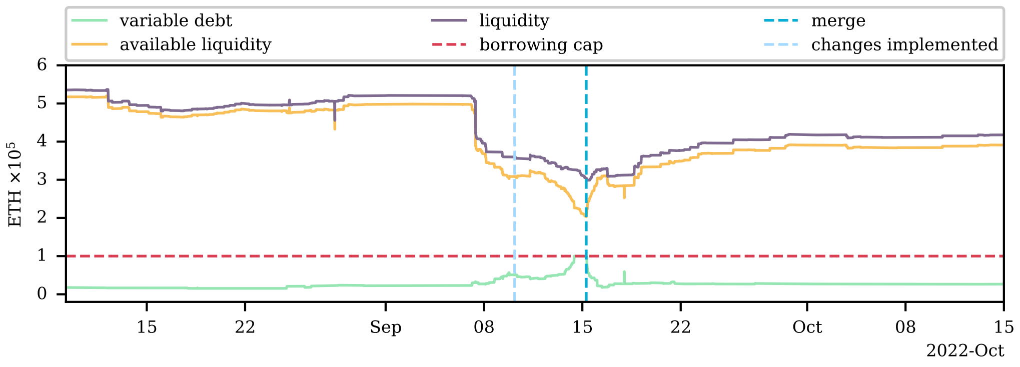
	U_{optimal}	r_0	r_{slope_1}	r_{slope_2}	R
variable rate	80%	2%	20%	4910%	20%

To better understand the effect of the imposed changes and the Compound ETH market ahead of the merge in general, we plot the evolution of the ETH debt and liquidity in Figure 2a. Further, we show the borrowing rate and utilization over time in Figure 2b. In both plots, we can clearly see the effect of the merge on Compound's ETH lending market.

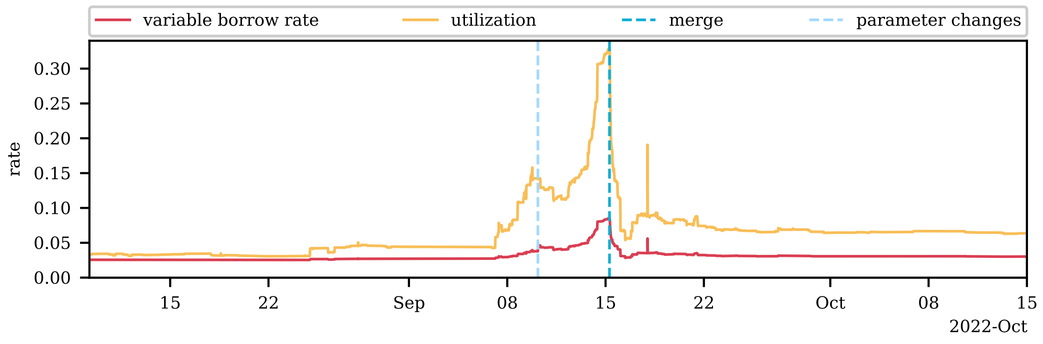
Notice that, as expected, two effects play out simultaneously ahead of the merge. For one, we observe an increase in debt, i.e., users appear to borrow ETH in order to be able to exploit this hard-fork-arbitrage opportunity. By borrowing ETH, they can increase the amount of ETH in their wallet and thereby increase the amount of ETHW they will receive once the chains forked. In fact, we looked at all wallets that increased their ETH debt by more than 1,000 ETH between 9 August 2022 and the merge. There were 18 addresses with a debt increase exceeding 1,000 ETH in total, and we could directly trace 50% of the borrowed funds to cryptocurrency exchanges (cf. Table 3).

When checking whether the addresses transferred the borrowed funds to cryptocurrency exchanges, e.g., Binance, Coinbase, FTX, etc., we monitor ETH(W) transfers from the wallets after they borrowed ETH from Compound. In particular, we check whether any funds were transferred to deposit addresses of exchanges. Exchanges typically have users transfer their assets to deposit addresses, these are created for each user, and then the exchanges forward these funds to their main addresses [45]. We search for transfers to exchanges by identifying the transfer of funds from the borrower's address to a deposit address that is then transferred to a known address of an exchange, in transaction data and their trace data on the Ethereum blockchain and EthereumPoW blockchain. We remark that we only filter for these direct transfers to exchanges and thus might miss some additional transfers, where the borrowing address first transferred the funds to another address they control. Further, some of the biggest borrowers ahead of the merge were smart contracts (identified in italics in Table 3) as opposed to externally owned wallets. It is unlikely that the borrowed funds are transferred directly from these smart contracts to exchanges, and we were also never able to identify such a transfer. However, while we might miss some additional transfers to exchanges, only filtering for direct transfers to exchanges allows us to confidently say that the funds were indeed transferred to exchanges by the borrowers and lets us avoid over-counting. Thus, providing us with a lower bound for the total amount transferred to exchanges.

Table 3 notes to which exchange(s) funds were transferred by each borrower. Further, we indicate whether the borrower transferred the funds to exchanges before (on the Ethereum blockchain) and/or after (on the EthereumPoW blockchain) the merge. Any transfer to exchanges before the merge was ETH as opposed to ETHW, but some exchanges announced ahead of time that they would give the users ETHW for the ETH held with them [42]. We further indicate in Table 3 whether the funds transferred to exchanges amounted to at least 50% and/or 99% of the debt taken on by the address ahead of the merge. Most addresses, especially if we disregard the smart contracts where our method does not identify transfers to exchanges, transferred at least half of their new debt to exchanges. While we cannot determine the exact purpose of these transactions, it is highly likely that they intended to sell ETHW. Interestingly, the wallet with the highest debt increase ahead of the merge belonged to the now infamous Alameda Research: the cryptocurrency trading firm that allegedly



(a) Amount of ETH variable debt, available liquidity, and liquidity on Compound around the merge. The available liquidity is the difference between liquidity and debt. Ahead of the merge, indicated in the plot, Compound imposed a borrowing cap of 100'000 ETH and adjusted the interest rate curve. We indicate the time of the aforementioned changes and show the imposed borrowing cap. Notice that the borrowing cap was reached ahead of the merge. Additionally, liquidity drops ahead of the merge.



(b) ETH borrow rate and utilization ahead of the merge. Notice the sharp increase in utilization and borrow rate ahead of the merge and the subsequent sharp drop. Yet, the borrow rate and utilization did not reach dramatic levels and stayed below 0.1 and 0.35 respectively.

Figure 2 Compound ETH market around the time of the merge. Figure 2a shows the market's debt and liquidity, while Figure 2b the borrowing interest rate and utilization.

traded FTX customer funds and lost them. This led to the bankruptcy of FTX in November 2022 [44]. We also want to highlight that we tracked 49,206 ETH(W) to exchanges from only the borrowers shown in Table 3 – more than 49% of the ETH debt on Compound (100,000 ETH) ahead of the merge. At the time of the merge, the transferred ETHW was worth more than 2 Mio US$. Note that while borrowers did have to pay interest, these expenses were far lower than the value of ETHW, as is shown in Appendix B (cf. Figure 13).

Besides observing an increase in debt, especially by the 18 addresses discussed previously, we also note a decrease in liquidity (cf. Figure 2a). The reasons behind this decrease are likely more complex. For one, lenders might fear a rise in utilization in Compound's ETH market. When utilization levels are very high, lenders can no longer withdraw their funds (liquidity risk). Additionally, lenders might also wish to receive ETHW on the forked chain. However, ahead of the merge, the official ETHW Twitter account recommended for funds be withdrawn from multiple DeFi pools [15], including Compound's ETH market, in order to ensure that they would receive ETHW. There was even talk about freezing DeFi contracts on the ETHW fork [18]. However, if utilization in Compound's ETH market was high,

Table 3 We analyze ETH(W) transfers to cryptocurrency exchanges of all addresses (contracts are in italics) whose net borrowing preceding the merge exceeded 1,000 ETH on Compound. Note that the volume column indicates the debt increase. For each wallet, we display the exchanges to which funds were transferred and whether this occurred before or after the merge. Further, we indicate whether the wallet transferred the equivalent in value, at least 50%, and/or at least 99% of that debt to exchanges.

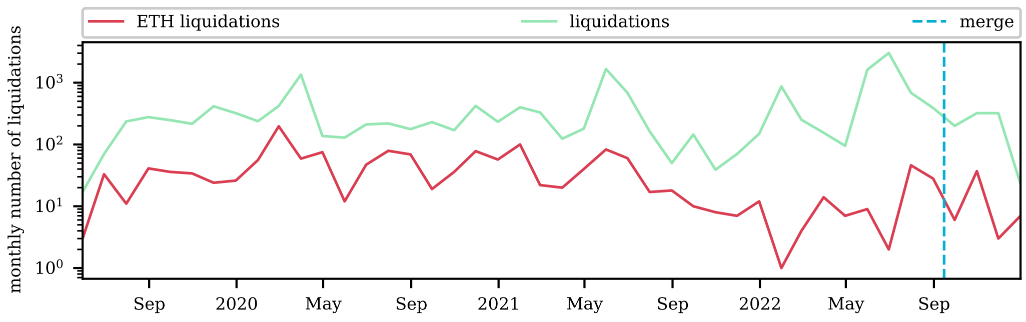
wallet address	volume [ETH]	exchanges	before	after	>50%	>99%
`0x712d0f306956a6a4b4f9319ad9b9de48c5345996`	15,000.00	FTX, OKX, MXC, Bybit	✓	✓	✓	✓
`0xa9f00c00ea5fd167da64917267e60f9d9430b321`	9,640.00	FTX	✓	✗	✓	✗
0xe40eea78752e969022c3dd18ae68713fd003e1c5	7,771.00					
`0xb0449ec1a8a60f95322617d6ed52e1ba1a7beb49`	7,000.05	FTX	✗	✓	✓	✓
0x8888882f8f843896699869179fb6e4f7e3b58888	6,611.92					
`0x66b870ddf78c975af5cd8edc6de25eca81791de1`	5,499.71	Binance, FTX, OKX, Bybit, MXC	✓	✓	✓	✓
`0xee8e0fcc8bff03ec5f100d02cb7b3196d78863a7`	4,499.92	FTX, MXC, Binance	✓	✓	✓	✓
`0x6a704a0e46dcc67a6316644372e261e8fb6f658c`	3,000.00		✗	✗	✗	✗
`0xcfc50541c3deaf725ce738ef87ace2ad778ba0c5`	2,498.00	Coinbase	✓	✗	✗	✗
`0x9681319f4e60dd165ca2432f30d91bb4dcfdfaa2`	2,000.00	FTX, Binance	✓	✓	✓	✓
`0x5add1cec842699d7d0eaea77632f92cf3f3ff8cf`	1,665.05	MXC	✗	✓	✗	✗
`0x42283fa21d5642c1744c2888f041ddea5d79149c`	1,650.00		✗	✗	✗	✗
`0xde6b2a06407575b98724818445178c1f5fd53361`	1,550.00	OKX	✗	✓	✓	✓
`0xb5c4402ff7cbe97785dddc768c4e3a4f033474fb`	1,501.00	FTX, MXC	✓	✓	✓	✗
`0xf71b335a1d9449c381d867f4172fc1bb3d2bfb7b`	1,400.00	FTX	✗	✓	✓	✓
`0x6d68c0f44e86587aa443ddb12ed9f10920195ada`	1,300.00	OKX, Bybit	✗	✓	✓	✓
`0xec97b52fc79f9ec7e951f050c80f65cc087197d3`	1,100.00		✗	✗	✗	✗
`0xe7072cdf38d3a6a4b92929abc302325f7b1ca628`	1,002.00		✗	✗	✗	✗

it is unlikely that it would decrease after the hard fork. Borrowers have no incentives to repay their ETHW debt on the EthereumPoW fork, as their collateral assets there are likely worthless. Thus, users would never be able to withdraw their ETH from Compound on the EthereumPoW chain even if they were not frozen. Despite a noticeable decrease in liquidity, the market's liquidity remained significantly larger than the protocol's debt, which reached the borrowing cap of 100,000 ahead of the merge.

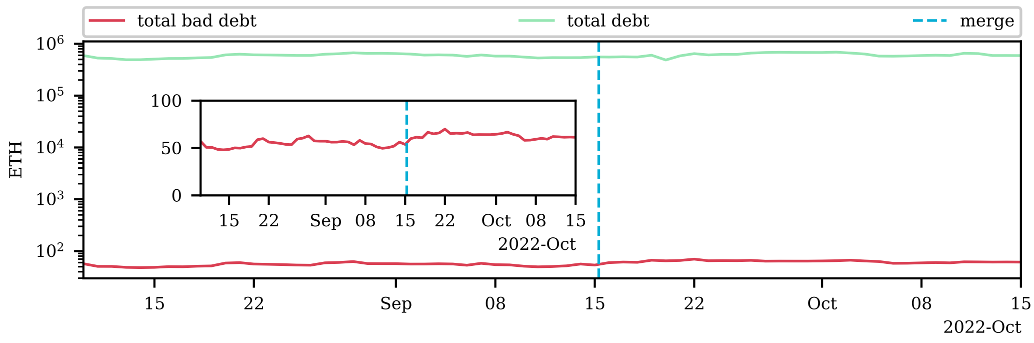
Thus, the intervention by the Compound community ensured that the protocol's utilization remained relatively low, i.e., it never exceeded 35% (cf. Figure 2b). As a consequence, the borrowing rate also remained relatively low. We further note that Compound did not experience an increase in liquidations of positions ahead of the merge. Figure 3a shows the total number of monthly liquidations and the share of those liquidations that had ETH debt covered by the liquidators. We presume that it was the intervention by the Compound community that helped prevent mass liquidation as the borrowing rate never exceeded 10%.

In addition to averting liquidation, the amount of bad debt on Compound, i.e., positions whose debt value exceeds the collateral value, did not increase significantly in the lead-up to the merge (cf. Figure 3a). Bad debt can be detrimental to a lending protocol, as the respective loans become irretrievable and present a loss for lenders. While generally, positions become liquidated before the debt value exceeds the collateral, extreme price swings can leave insufficient time to liquidate the positions. Yet, in this case, the stable amount of bad debt and the overall small share of bad debt (less than 0.01%) relative to the total debt on the protocol indicates that the increased rates did not impact the protocol's health significantly.

At the same time, the actions of the Compound community ensured that those who did manage to take out loans in time, i.e., before the borrowing cap was reached, could make significant profits. In Figure 4, we plot the break-even borrowing rate in the lead-up to the merge along with the actual Compound borrowing rate. The break-even borrowing rate at time t indicates the annualized rate a user would be willing to pay for borrowing ETH between time t and the merge, given the relative price between ETHW and ETH at time t.



(a) The monthly number of liquidations on Compound. We show in green the total number of liquidations, and in red the number of liquidations where liquidators covered the position's ETH debt.



(b) Total debt and bad debt, i.e., debt value exceeds the collateral value, on Compound before and after the merge. The bad debt is a very small proposition of the total debt – less than 0.01%. The inset shows a close-up of the bad debt on a linear scale. Notice that the bad debt on Compound does not spike up ahead of the merge.

Figure 3 Stability of ETH borrowing on Compound. Figure 3a shows the number of liquidations over time and Figure 3b shows the total and bad debt around the time of the merge. An increase in liquidations was avoided on Compound.

Note that we take the futures price at time t and not the price ETHW started trading at post-merge, as a borrower at time t could not know this price but rather had to rely on the price of future contracts. We compute the break-even borrow rate at time t as follows

$$\left(1 + \frac{p_{\text{ETHW}}(t)}{p_{\text{ETH}}(t)}\right)^{\frac{\Delta_{\text{year}}}{\Delta_{\text{merge}}}} - 1,$$

where $p_{\text{ETHW}}(t)$ is the price of ETHW in US\$, $p_{\text{ETH}}(t)$ is the price of ETH in US\$, Δ_{year} is the number of seconds in a year, and Δ_{merge} the number of seconds until the merge. Notice that at all times, the break-even borrow rate exceeded the actual borrow rate by at least one order of magnitude. Further, the break-even borrow rate was, for the most part, smaller than the maximum possible borrow rate on Compound. We show the maximum borrow rate before the imposed changes, $BR^1_{\max}$, in violet and the maximum borrow rate after the imposed changes, $BR^2_{\max}$, in light blue. Thus, users could infer that with a very high probability the short-term borrowing costs would be significantly lower than the value of the ETHW, they would receive once the chains forked. As a result, users were enticed to take leveraged long positions if they still could, i.e., until the borrowing cap was reached. Importantly, the opposite was true for lenders. They were insufficiently compensated and would have been better off withdrawing the funds and directly profiting from the merge.

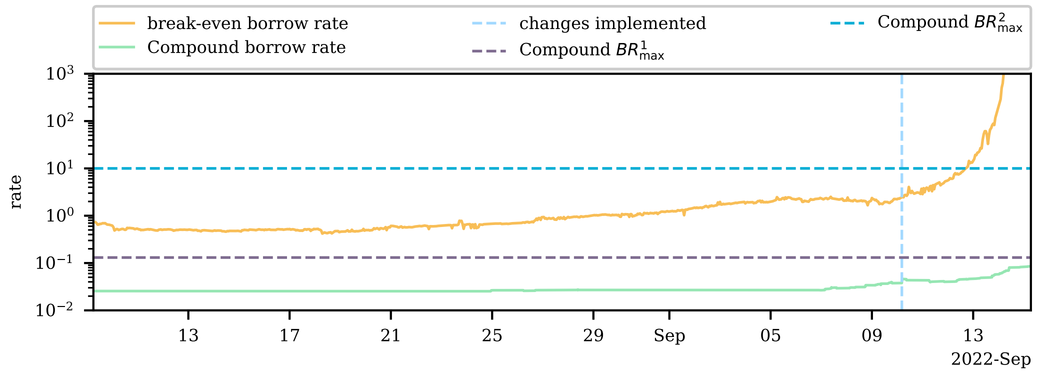


Figure 4 Break-even borrowing rate, i.e., the annualized interest rate users are willing to pay until the merge in return for ETHW, compared to Compound's borrowing rate. We indicate the maximum borrow rate before, BR^1_{max}, and after the implemented changes, BR^2_{max}. Note that Compound's borrowing rate was always at least one order of magnitude smaller than the break-even borrow rate.

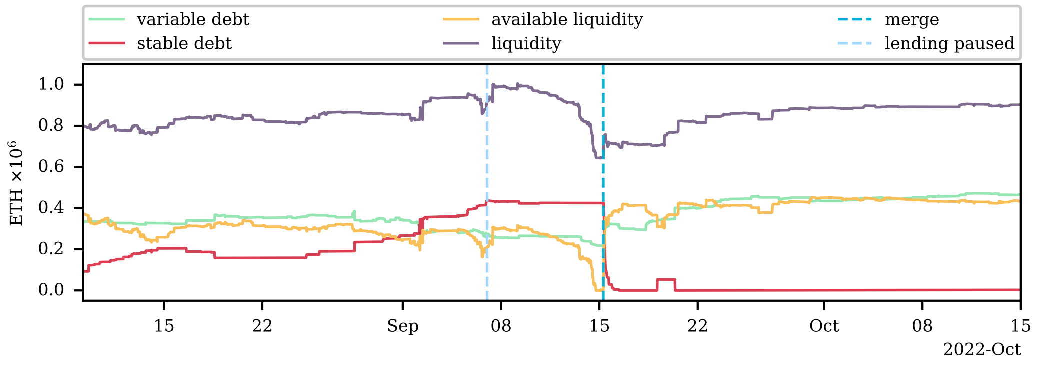
5.2 AAVE

The AAVE community started discussing the possible repercussions of the merge on 23 August 2022 [35]. They were also concerned that users would borrow as much ETH as possible to maximize their ETH holdings in anticipation of the fork. Such activity would increase utilization, make liquidations harder, and possibly lead to ETH suppliers withdrawing their ETH from the platform.

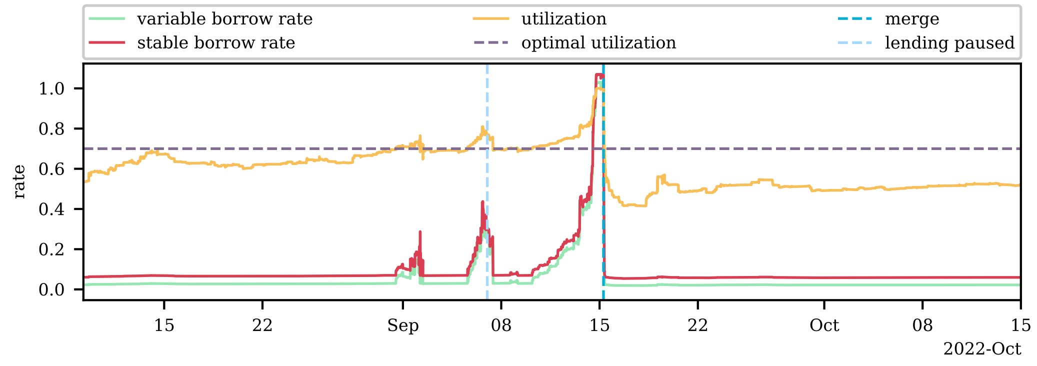
An additional challenge for AAVE, as opposed to Compound, was that they allow *staked ETH* (stETH) – the token users receive in exchange for staking their ETH with LIDO – as collateral. LIDO [28] is a protocol that allows you to easily stake your ETH on the PoS consensus layer – the Beacon chain. Normally, in order to stake ETH on the Beacon chain, users require 32 ETH, but LIDO is a liquid staking solution that allows its users to stake any amount. Thus, increasing accessibility to ETH staking. Staking rewards received by the ETH staked through LIDO on the Beacon chain are distributed to the users on a daily basis. More precisely, LIDO updates its Beacon chain balance every 24 hours on the Ethereum mainnet. The stETH balances in the wallets automatically update accordingly. Thus, stETH is an interest-earning token. stETH can be bought and sold by users.

Importantly, staking on the Beacon chain was activated more than a year ahead of the merge. Thus, many LIDO users had staked their ETH and received stETH in return ahead of the merge. Some stETH holders decided to utilize their stETH as collateral to take out ETH loans on AAVE, proceeded to stake the ETH they borrowed on LIDO, again received stETH, and continued this process. The staking rewards received by stETH holders historically exceeded the ETH borrowing rates on AAVE, making the aforementioned strategy profitable. The AAVE community feared liquidations of these positions as ETH borrowing rates were anticipated to rise ahead of the merge.

In their attempt to mitigate such scenarios, the AAVE community took a different route than the Compound community. They decided to pause all ETH lending on the platform ahead of the merge. A vote regarding the proposal was held between 2 and 6 September 2022 [26]. As the community was in favor of the changes, they were implemented on 7 September 2022 – a good week ahead of the merge. Thus, from 7 September onward, it was no longer possible to borrow ETH on AAVE.



(a) Amount of ETH variable debt, stable debt, available liquidity, and liquidity on AAVE around the merge. The available liquidity is the difference between liquidity and debt. Ahead of the merge AAVE paused ETH borrowing. We indicate the time of the aforementioned changes. Notice that the available liquidity continues to sink even though borrowing was paused as lenders are leaving the market. Furthermore, the slight increase in debt even after lending was paused is due to the fact that interest is accumulated on Aave.



(b) The ETH variable and stable borrow rate, as well as utilization ahead of the merge. Notice the sharp increase in utilization and borrow rate ahead of the merge and the subsequent sharp drop. The utilization of AAVE's ETH market reached 100% ahead of the merge even though lending was paused.

Figure 5 AAVE ETH market around the time of the merge. Figure 5a shows the market's debt and liquidity, while Figure 5b the borrowing interest rate and utilization.

As shown in Figure 5a these changes prevented a further increase in borrowing. In particular, the borrowed stable debt volume (red) increased until 7 September but remained basically flat from the implementation of these changes to the merge. Post-merge, the borrowed volume drops dramatically, highlighting that this borrowing activity was predominantly fuelled by speculators betting on the ETHW windfall. Thus, as plotted in Figure 5b, the implemented changes failed to keep borrowing rates in check. Liquidity providers enticed to directly hold ETH to themselves profit from the merge, withdrew their funds from the ETH pool, thus driving up the utilization rate. Additionally, the general uncertainty surrounding lending platforms in the weeks leading up to the merge may have spooked some lenders, further increasing outflows. We note that even with this intervention the utilization reached 100%, but the time it took to reach this level was likely prolonged. If the utilization had reached this level earlier, total interest payments would have been higher, thus compensating lenders more fairly.

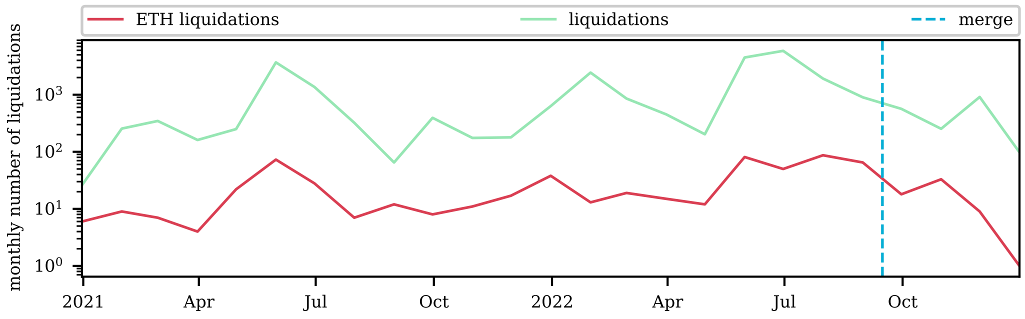
Table 4 We analyze ETH(W) transfers to cryptocurrency exchanges of all addresses (contracts are in italics) whose net borrowing in the lead-up to the merge exceeded 1,000 ETH on AAVE. Note that the volume column indicates the debt increase. For each wallet we display the exchange(s) to which funds were transferred and whether this occurred before or after the merge. Further, we indicate whether the wallet transferred the equivalent in value, at least 50%, and/or at least 99% of that debt to exchanges.

wallet address	**volume [ETH]**	**exchanges**	**before**	**after**	**>50%**	**>99%**
0xd275e5cb559d6dc236a5f8002a5f0b4c8e610701	49,998.89	Bitfinex	✓	✗	✓	✗
0x54dda22ae140edb605c73073eabb6f4aea2fc237	39,999.57	Binance, FTX	✓	✗	✓	✓
0xcde35b62c27d70b279cf7d0aa1212ffa9e938cef	22,762.39	OKX, FTX	✓	✓	✓	✓
0x236f233dbf78341d25fb0f1bd14cb2ba4b8a777c	17,500.00		✗	✗	✗	✗
0x68963dc7c28a36fcacb0b39ac2d807b0329b9c69	16,022.93	FTX	✓	✗	✓	✓
0xf6bf776c06a9946a7beba3bacbdaeb44e90684e1	15,000.00	FTX	✓	✗	✓	✗
0x68030330e8158be3fa5b3ec3c94bf07e42824b9b	14,894.68	Binance, Bybit, OKX, FTX	✓	✓	✓	✓
0x5beabefb832db8c0f5a2370b447613c8ebe572eb	8,951.73					
0x2bde0f6bfc26389fadccee7c1ca14bbf29c45812	7,520.00		✗	✗	✗	✗
0x4256886373b79e4e12c12b6796e99cde90f5f236	7,353.31					
0x321bf29f2d5dad204b5e25c31cac4348b6f29f96	7,304.10					
0x8d8b9c79196f32161bcb2a9728d274b3b45eb9af	7,051.00		✗	✗	✗	✗
0xe40eea78752e969022c3dd18ae68713fd003e1c5	6,950.00					
0x4c1bda12452146184a8085c890e22fb7933aff2f	6,250.00					
0x48608b596888e0b9512be7f3f5f2e05d3c3d5180	5,300.00	FTX	✓	✗	✓	✓
0xcc8a1601a32b48cebf45224ca6d786c24414a10b	4,500.00	FTX	✓	✗	✗	✗
0x2a0fe598e69a4fc882f6f7a954662cf0a0819467	4,460.18					
0xca00bf9fa7bee6034565bf5d8e7f95fe52182241	4,200.00		✗	✗	✗	✗
0x1bda63dab1743089af8c0c94ed0b75772a9b9858	4,000.40	Binance	✓	✗	✓	✓
0xadbab4f38ff9dcd71886f43b148bcad4a3081fb9	3,998.62	MXC	✗	✓	✗	✗
0x916792f7734089470de27297903bed8a4630b26d	3,768.00	FTX	✗	✓	✓	✓
0xe8b22a88deb45c7848d394fd039b8d811511a9f3	3,000.00	Binance, OKX, FTX	✓	✗	✓	✓
0x9b1945d5434b2e69eb00e44b9022ad4172922eb5	2,999.00					
0x2662d826a86d602c01affd6974432e43009eb14b	2,729.17					
0xb1473f4d2e416310e4715cc7bcbe8074aed24a56	2,200.00	Bybit, OKX	✗	✓	✓	✓
0xb5c4402ff7cbe97785dddc768c4e3a4f033474fb	2,180.00	MXC, FTX	✓	✓	✓	✗
0x3da0ca6c78ea283200a0d5b2790aa5de280e43cc	2,000.00		✗	✗	✗	✗
0x66b870ddf78c975af5cd8edc6de25eca81791de1	1,998.65	Binance, Bybit, FTX, MXC, OKX	✓	✓	✓	✓
0x1778767436111ec0adb10f9ba4f51a329d0e7770	1,711.01	FTX	✓	✗	✓	✓
0xa1175a219dac539f2291377f77afd786d20e5882	1,600.00		✗	✗	✗	✗
0x474e2cb1aac71f66d0aa7adb0cd92c919f842fe4	1,599.88	Binance, MXC, Bybit, FTX	✓	✓	✓	✓
0x7f960b97b12ef8b6828529e961f6646ad764d90b	1,500.00	MXC	✗	✓	✓	✓
0x307111465e4cedd89fa28b9768981b8768a3cabe	1,400.00					
0x09d0ed8d3ebf0b0b5d2a3d7096546d6d7085b8bb	1,364.00	FTX	✗	✓	✓	✓
0x74b8c7680502931c33d9446e26592b8318eb7248	1,110.99					
0x3d9663bbd7f238b940ad4244fac58ff54ce870dc	1,100.00	Binance, FTX	✓	✗	✓	✓
0xecfb36305daa4244281d8249783bddf0918db361	1,016.00	Binance, FTX	✓	✗	✓	✓
0x7ce450c2974746e3d21b13cb05d253e6fd56f6bd	1,000.00	OKX	✗	✓	✓	✓
0x05f65845a202aadabce5475b6495f54fb2073b04	1,000.00	Peatio	✓	✗	✓	✗

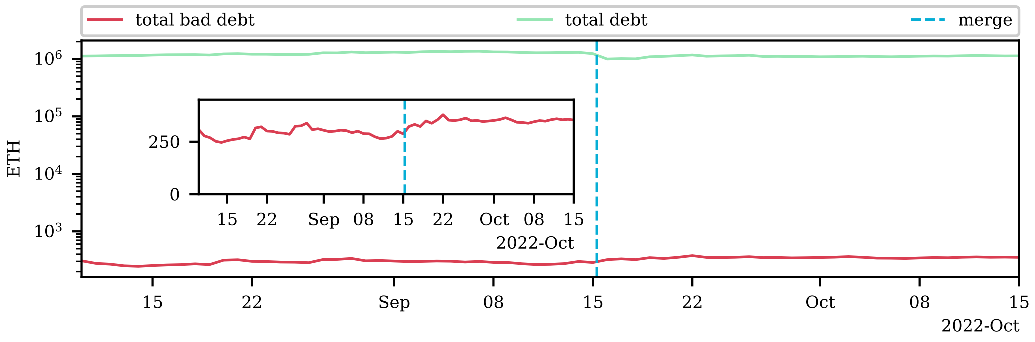
Similarly, as done for Compound in Section 5.1, we again track the funds borrowed by the biggest AAVE borrowers ahead of the merge to see whether the funds were transferred to cryptocurrency exchanges – indicating that the borrowers wanted to sell the ETHW. In Table 4, we indicate whether and when funds were transferred to exchanges for all 38 ETH borrowers on AAVE that increased their debt by more than 1,000 ETH ahead of the merge. We highlight smart contract borrowers in italics and note again that for these we were not able to track direct transfers to cryptocurrency exchanges. Focusing on the borrowers whose addresses were externally owned wallets, we find that more than 72% of those transferred at least 50% of the debt they took on directly to cryptocurrency exchanges, while 58% transferred at least 99%. Further, we find that these 29 borrowers moved 39% (251,329 ETH(W)) of all funds borrowed on AAVE (643,367 ETH(W)) to cryptocurrency exchanges. The transferred ETHW amounted to more than 11 Mio US$ at the time of the merge. The lenders were essentially deprived of these funds, as they were not fairly compensated for their service as liquidity providers. As we illustrate in Appendix B in Figure 14, the borrowing costs were far lower than the value of ETHW.

While the borrowing rates on AAVE increased significantly before the merge, the worst fears of mass liquidations did not materialize. As shown in Figure 6a, the number of liquidations occurring on AAVE did not rise significantly in the lead-up to the merge. Similarly, as plotted in Figure 6b, no significant increase in the proportion of bad debt can be observed. While the utilization rate did spike, this only persisted for a short time, and as the maximal borrowing rate is capped at 103% on an annualized basis, the total interest expense for borrowers was manageable.

In Figure 7, we again plot the break-even rate (yellow). As for Compound, the actual borrowing rates were significantly lower than the break-even rates, making leveraged long positions in ETH profitable. Given that ETHW futures traded at about 3%, the maximal annual borrowing rate of 103% that AAVE allows was orders of magnitude lower than the break-even rate. Thus, the protocol again inadequately compensated lenders.



(a) The monthly number of liquidations on AAVE. We show in green the total number of liquidations, and in red the number of liquidations where liquidators covered the position's ETH debt.



(b) Total and bad debt, i.e., debt value exceeds collateral value, on AAVE before and after the merge. The bad debt is a very small proposition of the total debt – less than 0.04%. Notice that the bad debt on AAVE does not spike up ahead of the merge.

Figure 6 Stability of ETH borrowing on Aave. Figure 6a shows the number of liquidations over time and Figure 6b shows the total and bad debt around the merge. As with Compound, an increase in liquidations was avoided.

Figure 8 visualizes the average size of a lending position over time. Observe the clear drop prior to the merge indicating that primarily larger liquidity providers exited, whereas the smaller players were more likely to stay put. We presume that the larger and likely

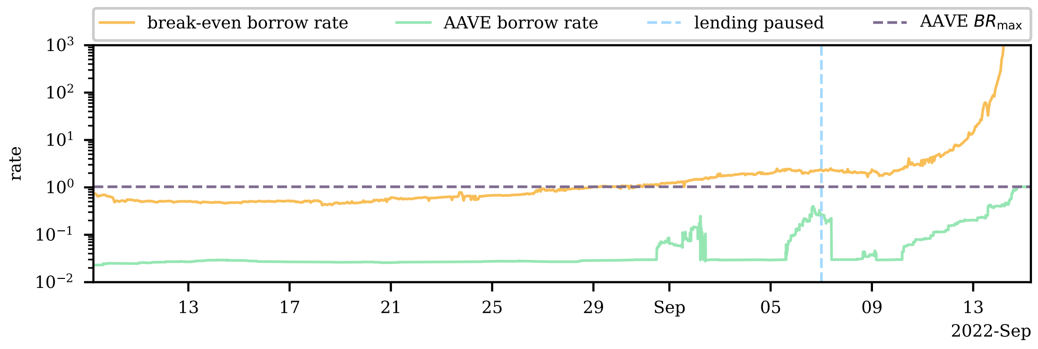


Figure 7 Break-even borrowing rate, i.e., the annualized interest rate users are willing to pay until the merge in return for ETHW, in comparison to variable borrow rate on AAVE. We further indicate the protocol's maximum borrow rate, $BR_{\max}$. Notice the significant discrepancy between the AAVE variable borrow rate and the break-even borrow rate.

more sophisticated liquidity providers exited the AAVE with their ETH in time, while the smaller and likely less sophisticated lender remained stuck in the pool once the utilization reached 100%. Thus, the smaller lenders bore the brunt of the losses as they missed out on the hard-fork arbitrage.

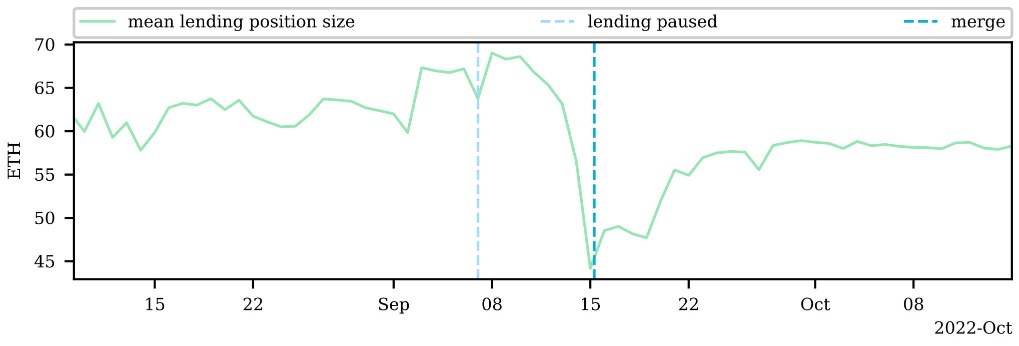


Figure 8 Mean size of the lending positions on AAVE as a function of time. Notice the drop prior to the merge, indicating that primarily large lenders exited, while smaller liquidity providers were left back.

Unlike Compound, AAVE allows stETH to be used as collateral. The value of stETH stems from the staking rewards as well as the fact that in the future stETH holders will be able to swap stETH for ETH. However, unlike current ETH holders, stETH owners did not receive ETHW after the merge. Therefore, unsurprisingly, stETH was trading at a discount to ETH in the months before the merge (cf. Figure 9). Observe that this discount is comparable to the value of ETHW. After the merge, the stETH-ETH price recovered and was again trading close to parity.

While AAVE does not facilitate stETH borrowing, users can use stETH as collateral to borrow ETH, which in turn can be used to acquire yet more stETH. AAVE allowed 1 stETH to be used to borrow 0.73 ETH – enabling a popular trading strategy as the difference between the staking rewards and the ETH borrowing costs was quite significant (cf. Figure 10). For months the staking rewards were significantly higher, enticing investors

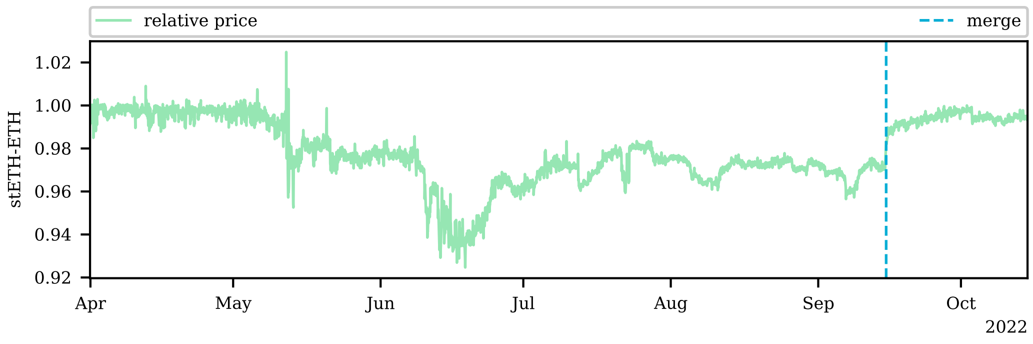


Figure 9 The stETH price over time. In the lead-up to the merge stETH was trading at a significant discount but returned to price parity after the merge.

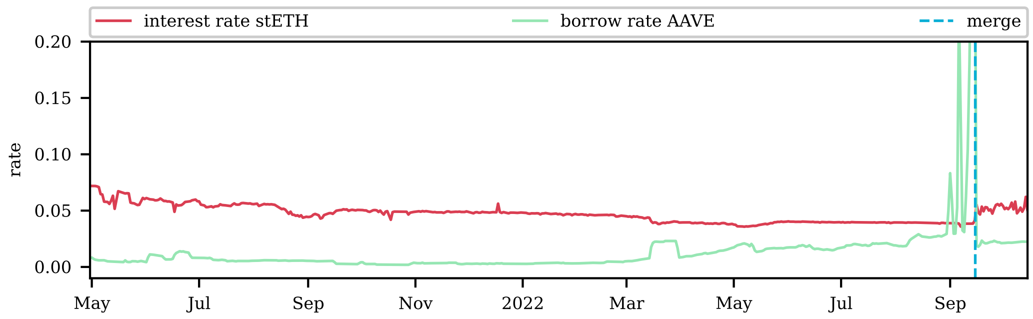


Figure 10 Annualized stETH staking rewards (red) and the ETH borrowing rate (green) on AAVE. In the months prior to the merge the staking rewards were significantly higher than the borrowing costs, making it profitable for traders to use leverage to buy stETH.

to take on leverage. However, this reversed in the lead-up to the merge as borrowing rates spiked. This is significant as we find 20.3% of the stETH market capitalization is deposited on AAVE (cf. Appendix A). As AAVE does not support stETH borrowing, there is no reason to deposit other than to take out loans. More than one-fifth of the stETH pledged as collateral makes mass liquidations a real threat to the stETH price and, thereby, to the blockchain consensus layer, as staking power can be acquired at a discount.

6 Discussion

Intervention by DAOs. While borrowing costs in the days prior to the merge were high relative to the rates typically seen on these protocols, they were still far lower than the payoff an ETH borrower could expect. Thus, rates were too low from the lender's perspective. Lenders were neither adequately compensated for forgoing this arbitrage opportunity nor for the uncertainty that surrounded DeFi in general and lending protocols in particular. A borrower wishing to borrow for the last block prior to the merge should have paid interest at least equal to the price of one ETHW token – orders of magnitudes more than the effective rate. Only such a rate would have compensated a lender for not profiting from the hard-fork-arbitrage. Furthermore, AAVE liquidity providers were unable to withdraw their funds as the utilization rate approached 100%, thus depriving them of even having the option to withdraw. Ultimately, the profits the arbitrageurs made at the expense of liquidity providers, whom the protocols failed to adequately compensate.

In comparison, the largest centralized crypto exchange, Binance, encouraged users to repay their ETH and, furthermore, withheld the ETHW that was awarded for borrowed ETH [43]. Whether this course of action was more equitable is surely debatable. In fact, Binance suspended withdrawals altogether [43]. However, it avoided the situation of charging extremely high borrowing rates but still kept a lending market open. This was particularly beneficial for traders who wanted to borrow ETH for reasons other than speculating on the ETHW tokens.

This discussion highlights that the merge was an extraordinary situation that led to interventions on both decentralized and centralized lending platforms. They capped borrowing, limited rates, or tried to deter speculators by withholding the ETHW tokens. For DeFi lending protocols, this meant having to give up their "no intervention" mantra. While these interventions safeguarded the protocols as a whole, they were paid for by the liquidity providers. This lack of compensation contributed to the DeFi lending market drying up. Given the continued rapid growth of DeFi and the particular importance these protocols play in this ever-more intertwined system, the lack of liquidity in such extraordinary market situations poses a grave threat to the broader Ethereum ecosystem. As shown during the 2008 financial crisis, the drying up of the lending market can greatly exacerbate market downturns.

Furthermore, we stress that the merge was announced well in advance leaving ample time for the protocols to come up with and implement their proposals. While our study focuses on one particular hard-fork, Ethereum has gone through more than a dozen hard-forks since its genesis [40]. Not all of these were announced far ahead of time. Thus, future more grave outcomes for lending protocols in the face of hard forks cannot be discounted. Additionally, we note that external market shocks are rarely as foreseeable as in this case. Rapid, unexpected developments could deprive DAOs of this course of action and pose greater threats to the viability and security of the protocol. For instance, the recent accumulation of bad debt on AAVE left by an attacker that borrowed a large amount of CRV tokens for short-selling, could not be prevented due to events unfolding much more rapidly [10].

Security concerns beyond lending protocols. We note the potential ramifications of the highly leveraged stETH positions the borrowing spirals created. The reversal of the difference between staking rewards and borrowing costs made these leveraged positions in stETH unprofitable. This posed a grave security threat, as a total of 20.3% of stETH was locked on AAVE. Liquidations due to rising borrowing costs and/or a falling stETH price would further devalue the collateral of other leveraged stETH holders, resulting in a downward spiral. The ramifications thereof would spill over to the wider DeFi ecosystem, as users could, for example, acquire stETH and its staking power at a significant discount. As LIDO accounts for more than a fourth of staking power on the Beacon chain [34] and given the size of these lending protocols, their viability is crucial to DeFi as a whole.

Market inefficiencies. Finally, we add that the ETH lending market on AAVE and Compound is an example of market inefficiencies in the cryptocurrency market. In an efficient market, i.e., a market where prices reflect all relevant information [29], the hard-fork-arbitrage we studied in this work should not exist as the combined market value of ETH and ETHW after the fork should be equal to the market value of ETH before the fork (no arbitrage condition). Thus, the hard-fork-arbitrage is an empirical example of market inefficiencies in DeFi.

7 Conclusions

Given the central role of lending protocols in DeFi and the composability of the latter, the stability of these protocols is crucial to the entire ecosystem. Therefore, unsurprisingly, concern grew in the months leading up to the merge that hard-fork-arbitrageurs would borrow large amounts of ETH, which would drive up rates and potentially lead to mass liquidation. Both Compound and AAVE saw no alternative to intervention and effectively capped borrowing.

Our analysis finds that these interventions may have helped prevent widespread liquidations. However, these interventions led to market distortions and were made at the expense of the protocol's liquidity providers. On the other hand, large borrowers like Alameda Research, who speculated on the hard-fork-arbitrage transferred proceeds from the hard-fork arbitrage worth more than 13 Mio US$ at the time of the merge to centralized exchanges. These tokens were in effect extracted from the liquidity providers, who were by far not fairly compensated for either their service or for the risk they bore. Furthermore, as the utilization rate approached 100%, the lending market ceased to function. Neither could liquidity providers withdraw nor could new debt be taken on, effectively drying up the DeFi lending market.

Finally, we find that the increased complexity resulting from the ever-increasing composability of DeFi poses security concerns not only for DeFi protocols but even for the consensus layer. For example, over one-fifth of the ETH staked through LIDO was locked on AAVE as collateral during the merge. Widespread liquidations would have led to a dramatic drop in the price of stETH, effectively giving a discount to anyone wishing to acquire staking power.

References

1 Aave. `https://aave.com/`, 2022.

2 Sirio Aramonte, Sebastian Doerr, Wenqian Huang, Andreas Schrimpf, et al. Defi lending: intermediation without information? Technical report, Bank for International Settlements, 2022.

3 Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch Lafuente. Sok: Lending pools in decentralized finance. In *International Conference on Financial Cryptography and Data Security*, 2021.

4 Chainlink. Decentralized data feeds. `https://data.chain.link/`, 2023.

5 Jonathan Chiu, Emre Ozdenoren, Kathy Yuan, and Shengxing Zhang. On the inherent fragility of defi lending. `https://www.snb.ch/n/mmr/reference/sem_2022_06_03_chiu/source/sem_2022_06_03_chiu.n.pdf`, 2022.

6 Coinmarketcap: Today's cryptocurrency prices by market cap. `https://coinmarketcap.com/`, 2023.

7 Compound. `https://compound.finance`, 2022.

8 Contract 0x0c3f8df27e1a00b47653fde878d68d35f00714c0. `https://etherscan.io/address/0x0C3F8Df27e1A00b47653fDE878D68D35F00714C0#readContract`, 2023.

9 Michael Darlin, Georgios Palaiokrassas, and Leandros Tassiulas. Debt-financed collateral and stability risks in the defi ecosystem. In *2022 4th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 5–12. IEEE, 2022.

10 EigenPhi. How aave's $1.6 million bad debt was created, 2022. URL: `https://medium.com/@eigenphi/how-aaves-1-6-million-bad-debt-was-created-915898d466cc`.

11 Shayan Eskandari, Mehdi Salehi, Wanyun Catherine Gu, and Jeremy Clark. Sok: Oracles from the ground truth to market manipulation. *arXiv preprint arXiv:2106.00667*, 2021.

12 Ethereum classic. `https://ethereumclassic.org/`, 2023.

13 Ethereum Foundation. The merge. `https://ethereum.org/en/upgrades/merge/`, 2023.

14 Ethereumfair. https://etherfair.org/, 2023.

15 EthereumPoW. Ethereumpow (ethw) official twitter, 2022. URL: https://twitter.com/EthereumPoW/status/1560044879920607233?s=20&t=Kzk1R9j4BK47WKu32M-3zw.

16 Ethereumpow. https://ethereumpow.org/, 2023.

17 ethereumpow. Ethereumpow (ethw) official. https://github.com/ethereumpow, 2023.

18 Ethpow params, 2022. URL: https://github.com/Paul286/go-ethereum/blob/7cda05854f4db007f459f0771ab24f9450296a89/params/ethpow_params.go.

19 ETHW Explorer. https://www.oklink.com/en/ethw/, 2023.

20 Emilio Frangella. Aave borrowing rates upgraded. https://medium.com/aave/aave-borrowing-rates-upgraded-f6c8b27973a7, 2022.

21 Robin Fritsch, Marino Müller, and Roger Wattenhofer. Analyzing voting power in decentralized governance: Who controls daos? *arXiv preprint arXiv:2204.01176*, 2022.

22 Lewis Gudgeon, Sam Werner, Daniel Perez, and William J Knottenbelt. Defi protocols for loanable funds: Interest rates, liquidity and market efficiency. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 92–112, 2020.

23 Stefan Kitzler, Friedhelm Victor, Pietro Saggese, and Bernhard Haslhofer. Disentangling decentralized finance (defi) compositions. *arXiv preprint arXiv:2111.11933*, 2021.

24 Label Word Cloud. https://etherscan.io/labelcloud, 2023.

25 ledgerwatch. Erigon. https://github.com/ledgerwatch/erigon, 2023.

26 Paul Lei, Jonathan Reem, Nick Cannon, Watson Fu, Tony Salvatore, and Sarah Chen. Pause eth borrowing. https://app.aave.com/governance/proposal/97/, 2022.

27 Lending tvl rankings. https://defillama.com/protocols/lending/Ethereum, 2023.

28 Lido. https://lido.fi, 2022.

29 Burton G Malkiel. Is the stock market efficient? *Science*, 243(4896):1313–1318, 1989.

30 monet supply. Proposal: Adjust eth interest rate model. https://www.comp.xyz/t/proposal-adjust-eth-interest-rate-model/3493/1, 2022.

31 MonetSupply. ceth risk mitigation. https://compound.finance/governance/proposals/122, 2022.

32 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

33 Daniel Perez, Sam M Werner, Jiahua Xu, and Benjamin Livshits. Liquidations: Defi on a knife-edge. In *International Conference on Financial Cryptography and Data Security*, pages 457–476. Springer, 2021.

34 Pool distribution, 2023. URL: https://beaconcha.in/pools.

35 Primoz. [arc] aave ethpow fork risk mitigation plan. https://governance.aave.com/t/arc-aave-ethpow-fork-risk-mitigation-plan/9438, 2022.

36 Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An empirical study of defi liquidations: Incentives, risks, and instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 336–350, 2021.

37 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? *arXiv preprint arXiv:2101.05511*, 2021.

38 Joao Reginatto. Usdc and ethereum's upcoming merge. https://www.circle.com/blog/usdc-and-ethereums-upcoming-merge, 2022.

39 Tether. Usdt supports eth proof-of-stake transition. https://tether.to/en/usdt-supports-eth-proof-of-stake-transition/, 2022.

40 The history of ethereum. https://ethereum.org/en/history/, 2023.

41 Palina Tolmach, Yi Li, Shang-Wei Lin, and Yang Liu. Formal analysis of composable defi protocols. In *International Conference on Financial Cryptography and Data Security*, pages 149–161. Springer, 2021.

42 Updates regarding the upcoming ethereum merge (2022-09-13). https://www.binance.com/en/support/announcement/notice-regarding-the-upcoming-ethereum-merge-2022-08-25-205de0801a4741c98270fdea6cb06697, 2022.

43 Updates regarding the upcoming ethereum merge (2022-09-13). `https://www.binance.com/en/support/announcement/updates-regarding-the-upcoming-ethereum-merge-2022-09-13-9a4805dffb8741a78f26075762a22a9c`, 2022.
44 Fran Velasquez. On-chain data shows close ties between ftx and alameda were there from the start: Nansen, 2022. URL: `https://www.coindesk.com/business/2022/11/22/on-chain-data-shows-close-ties-between-ftx-and-alameda-were-there-from-the-start-nansen/`.
45 Friedhelm Victor. Address clustering heuristics for ethereum. In *International conference on financial cryptography and data security*, pages 617–633. Springer, 2020.
46 Victor von Wachter, Johannes Rude Jensen, and Omri Ross. Measuring asset composability as a proxy for defi integration. In *International Conference on Financial Cryptography and Data Security*, pages 109–114. Springer, 2021.
47 web3.eth api. `https://web3py.readthedocs.io/en/v5/web3.eth.html`, 2023.
48 Jiahua Xu and Nikhil Vadgama. From banks to defi: the evolution of the lending market. In *Enabling the Internet of Value*, pages 53–66. Springer, 2022.
49 Yahoo finance. `https://finance.yahoo.com/`, 2023.
50 Aviv Yaish, Saar Tochner, and Aviv Zohar. Blockchain stretching & squeezing: Manipulating time for your best interest. In *Proceedings of the 23rd ACM Conference on Economics and Computation*, pages 65–88, 2022.
51 yfinance. `https://pypi.org/project/yfinance/`, 2023.

A stETH Market Capitalization and AAVE stETH Collateral

We plot the market capitalization of stETH before and after the merge in Figure 11. The stETH market capitalization corresponds to the combined ETH balance of LIDO validators on the Beacon chain. We point out that the stETH market capitalization is ever-increasing in that time frame, as withdrawals from the Beacon chain have not been activated. At the same time additional ETH is staked on the Beacon chain and the staked ETH balance increases as the validators are receiving rewards for performing their duties. Note that validators that do not carry out their duties correctly will be slashed and the ETH balance will reduce, this, however, did not happen to a large extent for LIDO validators.

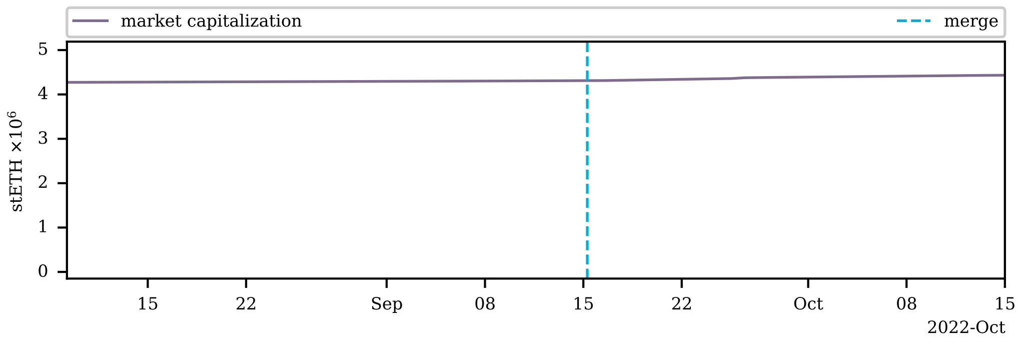


Figure 11 Market capitalization of stETH.

We also plot the amount of stETH locked on AAVE in Figure 12. As stETH cannot be borrowed, the stETH on AAVE is likely used as collateral to take out debt. A popular strategy for stETH holders was to take out ETH debt to be able to stake additional ETH with LIDO [35]. Thus, it is both astonishing and worrying at the same time that around 20% of all stETH are locked on AAVE. A price drop of stETH cloud cause liquidations of

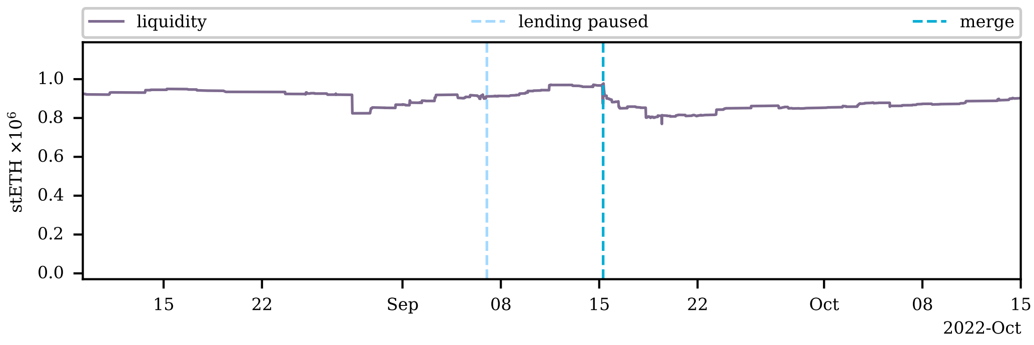


Figure 12 Amount of stETH locked on AAVE around the merge.

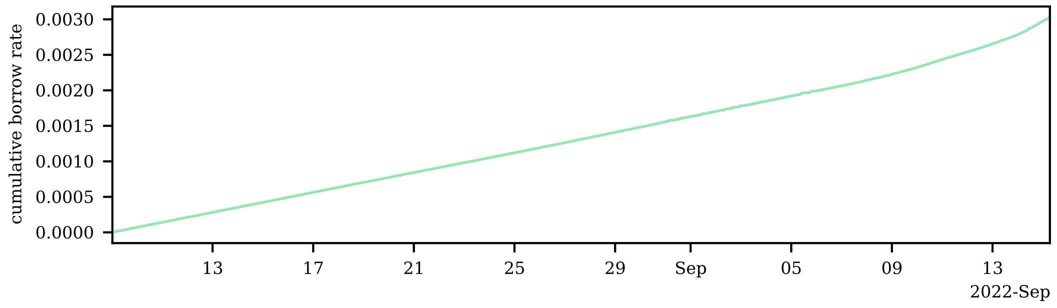


Figure 13 The cumulative interest rate for a borrower on Compound starting from 9 August 2022 until the merge.

loans on AAVE with stETH collateral which would further apply downward pressure on the stETH price. Thus, the high levels of stETH locked on AAVE pose a security concern for the Ethereum consensus layer.

B Cumulative Rates on AAVE and Compound

We plot the cumulative borrowing rate a borrower would have paid for an ETH debt taken out on 9 August 2022, the day ETHW started trading, and held until the merge, 15 September 2022. In Figure 13, we show the cumulative rate that would have been paid by an ETH borrower. A borrower would have paid around 0.03% for an ETH debt held for that time window – significantly less than the relative value of ETHW compared to ETH during the merge. Notice that the rate increases almost linearly as a consequence of the relatively low borrowing rate on Compound in the lead-up to the merge.

In Figure 14, on the other hand, we plot the cumulative borrowing rate for an ETH borrower on AAVE during the same time period. The cumulative rate paid on AAVE would have been higher than on Compound with 1%, but still significantly less than the price of ETHW in terms of ETH during the merge. The cumulative borrowing rate on AAVE rapidly increased starting from 7 September 2022 as a result of the sharp increase in the borrowing rate ahead of the merge.

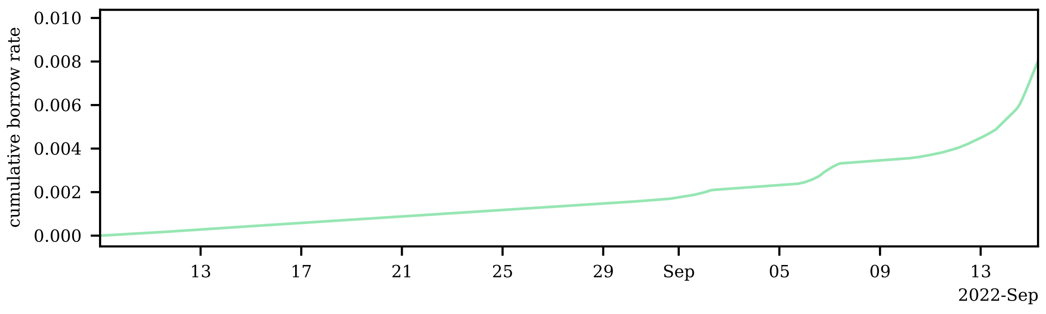


Figure 14 The cumulative interest rate for a borrower on AAVE starting from 9 August 2022 until the merge.

C Interest Rate Curves

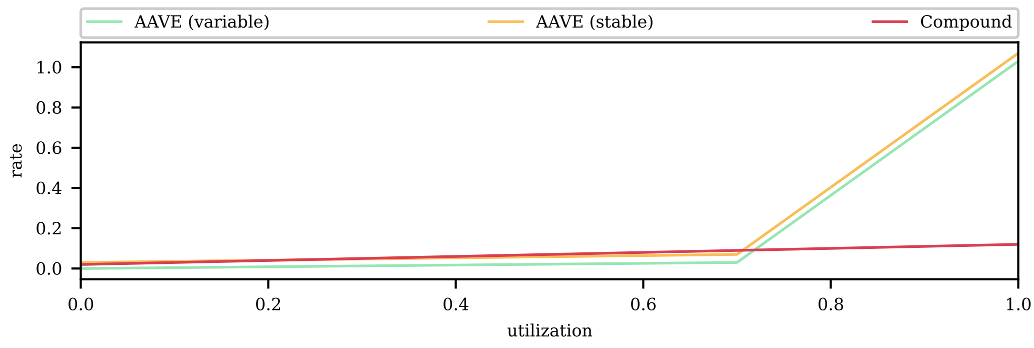


Figure 15 ETH borrowing rates on AAVE and Compound as a function of the utilization.

The interest rate for the two lending protocols are given in Section 3.3. Here, we show the interest rate curves as a function of utilization for AAVE and Compound in Figure 15. Furthermore, as described in Section 5.1, Compound updated their interest rate model in anticipation of the merge to the jump interest rate model that qualitatively looks similar to that of AAVE.

FairPoS: Input Fairness in Permissionless Consensus

James Hsin-yu Chiang
Aarhus University, Denmark

Bernardo David
IT University of Copenhagen, Denmark

Ittay Eyal
Technion, Haifa, Israel

Tiantian Gong
Purdue University, West Lafayette, IN, USA

Abstract

In permissionless consensus, the ordering of transactions or inputs in each block is freely determined by an anonymously elected block leader. A rational block leader will choose an ordering of inputs that maximizes financial gain; the emergence of automatic market makers in decentralized finance enables the block leader to front-run honest trade orders by injecting its own inputs prior to and after honest trades. Front-running is rampant in decentralized finance and reduces the utility of the system by extracting financial value from honest trades and increasing demand for block-space. Current proposals to prevent input order attacks by encrypting user inputs are not permissionless, as they rely on small static committees to perform distributed key generation and threshold decryption. Such committees require party authentication, knowledge of the number of participating parties or do not permit player replaceability and are therefore not permissionless. Moreover, alternative solutions based on sequencing inputs in order of their arrival cannot prevent front-running in an unauthenticated peer-2-peer network where message arrival is adversarially controlled.

We present *FairPoS*, the first consensus protocol to achieve input fairness in the permissionless setting with security against adaptive adversaries in semi-synchronous networks. In FairPoS, the adversary cannot learn the plaintext of any client input *before* it is included in a block in the chain's common-prefix. Thus, input ordering attacks that depend on observing pending client inputs in the clear are no longer possible. In FairPoS, this is achieved via Delay Encryption (DeFeo *et al.*, EUROCRYPT 2021), a recent cryptographic primitive related to time-lock puzzles, allowing *all* client inputs in a given round to be encrypted under a key that can only be extracted after enough time has elapsed. In contrast to alternative approaches, the key extraction task in delay encryption can, in principle, be performed by any party in the permissionless setting and requires no distribution of secret key material amongst authenticated parties. However, key extraction requires highly specialized hardware in practice. Thus, FairPoS requires resource-rich staking parties to insert extracted keys into blocks, enabling light-clients to decrypt past inputs and relieving parties who join the execution from decrypting all inputs in the entire chain history. Realizing this in proof-of-stake is non-trivial; naive application of key extraction to proof-of-stake can result in chain stalls lasting the entire key extraction period. We overcome this challenge with a novel *key extraction protocol*, which tolerates adversarial delays in block delivery intended to prevent key extraction from completing on schedule. Critically, this also enables the adoption of a new *longest-extendable-chain* rule which allows FairPoS to achieve the same guarantees as Ouroborous Praos against an adaptive adversary.

2012 ACM Subject Classification Security and privacy → Privacy-preserving protocols

Keywords and phrases Front-running, Delay Encryption, Proof-of-Stake, Blockchain

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.10

Related Version *Full Version*: `https://eprint.iacr.org/2022/1442`

Funding *James Hsin-yu Chiang*: The author's work was partially conducted during a research visit at the Israel Institute of Technology (Technion), and was supported by the Otto Mønsted foundation, Innovation Centre Denmark and DTU Compute.
Bernardo David: The author's work was supported by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B.
Ittay Eyal: The author's work was supported in part by the Israel Science Foundation (grant No. 1641/18), Avalanche Foundation and IC3.
Tiantian Gong: The author's work was supported by the National Science Foundation (NSF) under grant CNS-1846316 and an unrestricted gift from Supra Research.

1 Introduction

In permissionless consensus, the ordering privilege of the block leader is exploited in front-running [18], where adversarial inputs can be interleaved with honest inputs to extract financial value from the honest victim in applications such as automatic market makers [5]. Such behaviour financially penalizes the honest user, but also generates excess demand for block-space since front-running attacks [5] always require additional inputs from the adversary, inflicting block congestion at times, as acutely observed on Avalanche [2]. Current proposals to mitigate front-running with varying notions of input fairness violate assumptions underlying permissionless consensus such as Proof-of-Stake (PoS) [19].

A commonly proposed notion of input fairness requires encrypting inputs which are then decrypted after finalization[1]. To guarantee timely decryption and to avoid malicious parties withholding the reveal of the plaintext input, *threshold decryption* [32, 7] or *identity-based encryption* [33, 21] involving static committees have been proposed. The honest-majority committees with distributed private or master key material then guarantee the decryption of inputs as specified by the protocol. However, such protocols require authenticated parties to prevent Sybil interference and assume secure, private communication and are therefore not permissionless; ongoing research efforts to lift such protocols into the permissionless setting are discussed in Section 2.

Alternatively, the notion of sequencing transactions in the order of their arrival at honest consensus nodes has been proposed in [29, 28, 27, 13]. This is meaningful in the permissioned setting where communication between client and consensus node is fast, preventing an adversary to observe a pending transaction and then emit a front-running transaction which can then propagate faster than the victim's transaction. However, this notion of input order fairness does not easily translate to the permissionless setting, where transactions are propagated across a permissionless, unauthenticated peer-to-peer network where delay is adversarially controlled.

We introduce FairPoS, a PoS blockchain consensus protocol that achieves a novel notion of *input fairness* (Def. 6, Thm. 19) in permissionless consensus, while retaining the security guarantees of Ouroborous Praos [19]. As in Praos, we prove security against an adaptive adversary, which controls the network delay and corrupts parties as the protocol execution unfolds. Our novel notion of input fairness in permissionless consensus guarantees that the plain-text content of any finalized input (in the common-prefix) could not have been observed by the adversary prior to its finalization. FairPoS achieves this by encrypting inputs with the *delay encryption* scheme by DeFeo et al. [12], which improves on time-lock puzzles [34].

[1] In permissionless, longest-chain consensus, input finalization occurs when the block containing the input joins the common-prefix.

Classical time-lock puzzles store plaintext messages that can be obtained by clients after an *extraction process* that requires performing a known number of non-parallelizable sequential operations (*i.e.* requiring a certain minimum amount of time for recovering a message). However, naively encrypting inputs with time-lock puzzles requires a dedicated extraction process for each client input, which quickly becomes infeasible at higher throughput. Delay encryption, in contrast, allows all inputs in a block to be encrypted under a single unknown key, which can be extracted as time elapses. Hence, only a single key extraction is required for each block. The *extraction* procedure to recover the decryption key is parameterized to run in at least time d, and can be performed by any party with access to specialized hardware to ensure timely execution. This preserves adaptive security, as no relevant key material is learned upon corruption of an honest party.

Still, it is not practical for non-staking parties or clients with limited resources to perform key extraction. First, we expect only resource-rich participants to have access to the specialized hardware [1] necessary to perform extractions in d time; otherwise, a long-running, non-trivial extraction cost would be imposed on clients following the blockchain and interacting with smart contract applications. Secondly, without any integrated mechanism to publicly expose extraction keys, any party joining the protocol would need to perform key extractions for all blocks beginning from genesis, which becomes rapidly more expensive at higher chain lengths. A key contribution of FairPoS is a novel *key extraction protocol*, requiring staking parties to insert the extracted keys from past blocks into later, child blocks of the same chain within a fixed schedule, thus ensuring decryption keys are made public in lock-step with chain growth. The challenge here is to prevent the arbitrary delay of adversarial blocks to impede chain growth if honest parties cannot finish key extractions on time due to delayed arrival of past blocks. In standard blockchain consensus, chains in the local view can immediately be extended, but parties in FairPoS can only extend a chain if past key extractions are completed *on time*. Note that it is not sufficient to require a block to arrive at an honest party within the maximum network delay, so that key extraction can begin as intended. The adversary can trivially deliver dishonest blocks to a subset of honest parties only with the maximum permitted *receipt delay*, and then induce an additional network delay as this dishonest block is relayed to others. The local receipt delay of this block at other parties must then exceed the limit, causing irreconcilable inconsistencies and potential chain stall. Although such attacks cannot be prevented as the adversary is permitted to delay messages up to a maximum bound, they are carefully addressed in FairPoS by incrementing receipt delay bounds for blocks which are further away from the chain tip (Fig. 3). The novel *longest-extendable-chain* rule then asserts this notion of timeliness of block arrivals, guaranteeing that any honest chain can be extended by another honest leader within a fixed time. We highlight honest chain growth as a critical property and formally prove that FairPoS achieves both input fairness whilst maintaining the security of Praos [19].

Paper overview. We provide an overview of related work in Section 2. In Section 3, we introduce Delay Encryption and an abstract model of Ouroborous Praos execution (δ-PoS). In Section 4, we then define our proposed notion of Input Fairness for permissionless consensus and present the FairPoS model, extending δ-PoS with delay encryption and a novel "longest-extendable-chain" selection rule. In Section 5, we gently introduce FairPoS and formally demonstrate that achieves input fairness whilst maintaining the asymptotic security of Ouroborous Praos (δ-PoS) against an adaptive adversary. Full proofs of stated theorems and lemmas are provided in Appendix C of [16].

2 Related Work

Encrypt-and-reveal with timed cryptography. The basic idea of encrypt-and-reveal is to encrypt client inputs, and then to decrypt these when they are finalized in the blockchain. The "blockchain state" implied by the ordering of inputs is then "revealed" as past inputs are decrypted over time. For "decryption" to be permissionless, it must be possible without the knowledge of any secret key material; here, time-lock puzzles [34] were first proposed, which permit any party with the ciphertext to compute the decryption key with d squarings in a group of unknown order. Similarly, timed commitments were proposed in [11], which are accompanied by zero-knowledge proof of well-formedness; namely, that the time-lock commitment is well-formed and opens after d squaring operations. Time-lock puzzles have been formalized in the universal composability framework in Tardis [6], permitting secure composition with other protocols.

Mitigating front-running with time-lock puzzles in exchanges has been proposed in Clockwork [17]. In [26], blockchain inputs are encrypted with time-lock puzzles to prevent adversarial ordering based on the input plaintext. We note that this approach requires extracting decryption keys for each submitted input individually, which quickly becomes impractical for higher throughput levels. Open square [35] proposes a service which permits users to outsource the extraction of cryptographic time-locks to anonymous servers. De Feo et al. introduce the first time-lock primitive which permits all clients to encypt to the same session key, called Delay Encryption (§3.1), which improves on time-lock puzzles and minimizes "wasted" work spent on solving individual time-locks separately; thus, delay encryption represents the state-of-the-art in time-lock encryption.

In all timed-crypto primitives, the existence of a "fastest" hardware is assumed. Furthermore, it must be assumed that such hardware is accessible to all participants. We highlight two open challenges; firstly, there has been little research on parameterizing time-cryptography for real-world hardware designs. Secondly, the "fastest" hardware design is likely to be specialized and costly such that in practice, only resource-rich participants are likely to have access to such extraction hardware. A key objective of FairPoS is to address the second challenge by requiring block leaders to include extracted keys in the blockchain, thereby minimizing redundant work and allowing non-staking users to decrypt past inputs in lockstep with chain growth.

Encrypt-and-reveal with threshold cryptography. An alternative to timed-cryptography is to rely on a dedicated committee to generate a distributed master key, permitting the encryption and subsequent decryption of inputs, after they are finalized in the longest chain. Threshold decryption [7, 32] imposes a significant overhead, as each encrypted input must be individually decrypted by the dedicated committee. We note a recent you-only-speak-once (YOSO) line of work that investigates cryptographic protocols via anonymously elected committees [9, 24, 15, 14, 22] promising player replaceability in the absence of party authentication. Protocols in the YOSO model allow for permissionless solutions with the same effect of the aforementioned solutions based on threshold encryption. The concept of Encryption to the Future [14] allows for encrypting messages in such a way that they can be decrypted only by the block leader of a later slot, also allowing for realizing "Witness Encryption on Blockchains" (WEB) as proposed in [24] using threshold Identity Based Encryption (IBE). However, protocols in the YOSO model require techniques for proactively secret-sharing state to future committees; accomplishing this with high throughput in a practical manner remains an open problem. The central technique in the WEB construction from [14] is efficiently implemented in [33], which allows clients to encrypt their inputs to a *future round number* in string form; the identity-specific key associated with the current round number is then jointly released by the committee, permitting the public decryption of client inputs encrypted to the same round. An alternative construction called signature-based witness encryption is proposed by McFly [21], which realizes a special case of WEB with an efficient instantiation of the threshold IBE technique from [14]. This approach permits the encryption of an input to the set of committee verification keys and a round specific reference string. McFly scales to larger committee sizes, promising practicality for permissioned consensus systems. We emphasize that such encrypt-and-reveal approaches are not permissionless; an authenticated party must be assigned to each protocol role in an execution, communication between committee members must be secure and private and parties are not arbitrarily replaceable.

Fair ordering. A line of research from authenticated consensus [29, 28, 27, 13] proposes a notion of *fair input ordering.* A block leader will order inputs based on their order of arrival. However, fair ordering is only meaningful in a setting with a secure connection between client and round leaders: in an unauthenticated, peer-to-peer gossip network setting common in massively distributed permissionless blockchain protocols, the receipt-order of messages is adversarially controlled, making it difficult to justify any notion of fair message arrival. A secure connection with the next block leader implies public knowledge of its identity, contradicting the permissionless setting. The work of [27] lifts the notion of receipt-order-fairness to the permissionless setting; however, we argue the adopted notion of transaction arrival ordering is difficult to justify when client messages are propagated across an unauthenticated, peer-2-peer network with adversarially controlled delivery schedules; the adversary observing a propagating client transaction can inject their own inputs and control the receipt order for each honest consensus node. Moreover, this work is only secure against static adversaries.

3 Preliminaries

3.1 Delay Encryption

The delay encryption (DE) scheme by De Feo *et al.* [12] consists of the following four algorithms: A global DE.Setup parameterized with a security parameter $\lambda \in \{0,1\}^*$ and delay parameter d generates public encryption (DE.pk) and extraction (DE.ek) keys. In each round, a public session id $\in \{0,1\}^*$ is sampled, and DE.Encaps can be used to generate a pair (c, k) of a ciphertext c and a key k corresponding to id and the encryption key DE.pk. The DE.Extract algorithm runs in at least d time, and returns a session key idk, with which the DE.Decaps algorithm can compute a key k from ciphertext c for all (c, k) generated with the same session id and public paramaters from a given setup.

1. $\mathsf{DE.Setup}(\lambda, d) \rightarrow (\mathsf{DE.ek}, \mathsf{DE.pk})$
2. $\mathsf{DE.Encaps}(\mathsf{DE.pk}, \mathsf{id}) \rightarrow (c, k)$
3. $\mathsf{DE.Extract}(\mathsf{DE.ek}, \mathsf{id}) \rightarrow \mathsf{idk}$
4. $\mathsf{DE.Decaps}(\mathsf{DE.pk}, \mathsf{id}, \mathsf{idk}, c) \rightarrow k$

Delay encryption is an isogeny-based delay protocol, and similar to [20] is built from isogeny walks in graphs of pairing friendly supersingular elliptic curves. In implementations [20], such isogeny evaluations occupy memory space in the terabytes. Parties performing *Extract* are expected to deploy specialized FPGA hardware [1] in order to achieve the parameterized extraction time.

3.2 Longest-chain PoS model and security

We present a model of *longest-chain proof-of-stake* protocols, formalized by the Ouroborous line of work [31, 19, 3] and subsequent improvements [10, 30]. We adopt the approach of [31, 19, 3, 30], where the PoS protocol is modelled by two orthogonal components: the first describes the *leader election process* and the second part models the views of blockchain trees which result from a protocol execution *induced by a given leader schedule.*

Idealized leader elections. Time in PoS is divided into units named slots, each capturing the duration of a single protocol round. In a given round, a party with relative stake $\alpha \in (0,1]$ becomes a slot leader for a given slot with probability

$$\phi(\alpha) = 1 - (1-f)^{\alpha}$$

where parameter *active slot coefficient* f is the probability that a leader holding all stake will be elected leader in given slot: importantly, $\phi(\alpha)$ is maintained even if share α is split amongst multiple, virtual parties (eq. 2 in [19]). Let a characteristic string w be defined as a sequence of leader election results, where an election result at slot t is defined as follows.

$$w_t = \begin{cases} 0 & \text{a single honest leader} \\ 1 & \text{multiple honest leaders / adversarial leader} \\ \perp & \text{no leader} \end{cases}$$

In PoS [19], leader election is can be modelled by sampling characteristic strings from an idealized, *dominant distribution* $\mathcal{D}_{\alpha}^{f}$ that is strictly *more adversarial* than the *true* setting where the *adaptive adversary* corrupts up to $(1-\alpha)$ of the stake during the protocol execution (Theorem 8 in [19]). Thus, any security that holds in PoS executions induced by characteristic strings sampled from $\mathcal{D}_{\alpha}^{f}$ must also hold in the true protocol execution against an adaptive adversary dynamically corrupting up to $1-\alpha$ stake.

▶ **Definition 1** (**Dominant distribution $\mathcal{D}_{\alpha}^{f}$** (Definition 11 in [19])). *For an adaptive adversary corrupting up to $1-\alpha$ stake fraction and active slot coefficient $f \in [0,1)$, the dominant distribution $\mathcal{D}_{\alpha}^{f}$ is defined by the following probabilities:*

$$p_{\perp} = 1 - f \qquad p_0 = \phi(\alpha) \cdot (1-f) \qquad p_1 = 1 - p_{\perp} - p_0 \tag{1}$$

▶ **Definition 2** (**Blocks, chains, trees and branches**). *A block $B = (\mathsf{sl}, \mathsf{st}, \mathsf{d}, \mathsf{ldr})$ generated at slot sl contains state $\mathsf{st} \in \{0,1\}^*$, data $\mathsf{d} \in \{0,1\}^*$ and party ldr that generated B and was the elected block leader of slot sl. A chain is a sequence of blocks $B_0, ..., B_n$ associated with a strictly increasing sequence of slots, where B_0 is the genesis block, and the state of B_i is $H(B_{i-1})$, $H(\cdot)$ denoting a collision-resistant hash function. We write $\mathcal{C}.\mathsf{tip}$ to denote the block at the tip of chain $\mathcal{C}$ and $\mathcal{C}_j$ to denote a block $B \in \mathcal{C}$ such that $B.\mathsf{sl} = j$. If such a block does not exist, $\mathcal{C}_j = \perp$. Let $\mathcal{C}^{\lceil k}$ denote the chain obtained from $\mathcal{C}$ by removing the last k blocks. Multiple chains form a tree if their blocks share state. A branch $\mathcal{B}$ in a tree $\mathcal{T}$ is a chain which ends with a leaf block. We write $\mathcal{C} \preceq \mathcal{B}$ to indicate $\mathcal{C}$ is a prefix of $\mathcal{B}$. When quantifying over all chains in a tree, $\forall \mathcal{C} \in \mathcal{T}$, we quantify over all prefixes of all tree branches. Let $\mathsf{len}(\mathcal{C})$ denote the number of blocks in chain $\mathcal{C}$.*

A model of δ-PoS executions. As in [23, 31, 19, 3], we model the execution of PoS initiated upon the activation of an environment $\mathcal{Z}$, which spawns both honest parties $\mathcal{H}$ and an adversary $\mathcal{A}$. Upon each activation by the environment, each party executes the protocol

δ-PoS consensus model

The δ-PoS state $\Gamma_t = \left(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}\}_{i\in\mathcal{H}}, \mathcal{T}^{\mathcal{A}}\right)$ evolves in a single round $\Gamma_t \rightarrow^{w_{t+1}} \Gamma_{t+1}$ induced by environment $\mathcal{Z}$, adversary $\mathcal{A}$ and characteristic string w as follows.

Trees. Honest $\{\mathcal{T}^{(i)}\}_{i\in\mathcal{H}}$ and adv. $\mathcal{T}^{\mathcal{A}}$ evolve as follows:

T0 $\mathcal{T}_0^{(i)}$ consists of the genesis block.

T1 If $w_{t+1} = 0$, a single honest party runs **Extend** on the longest $\mathcal{C}$ in $\mathcal{T}^{(i)}$, which is then added to $\mathcal{T}^{(i)}$ and $\mathcal{T}^{\mathcal{A}}$.

T2 At any time during the round, adversary $\mathcal{A}$ may:

- **a)** Run **Extend** on a chain in $\mathcal{T}^{\mathcal{A}}$ for slot $t+1$ if $w_{t+1} = 1$, upon which the extended chain is added to $\mathcal{T}^{\mathcal{A}}$.
- **b)** Update any honest tree view $\mathcal{T}^{(i)}$ with an additional chain from $\mathcal{T}^{\mathcal{A}}$ or an honest message queue $\{\mathbf{m}^{(i)}\}_{i\in\mathcal{H}}$ (see **M2**).

Msgs. Honest message queues $\{\mathbf{m}^{(i)}\}_{i\in\mathcal{H}}$ evolve as follows:

M1 For chain $\mathcal{C}'$ extended by honest party i in the round (**T2**), the entry $(\mathcal{C}', \mathcal{H})$ is added to local message queue $\mathbf{m}^{(i)}$.

M2 At any time during the round, adversary $\mathcal{A}$ may deliver a chain from an entry $(\mathcal{C}, \mathcal{P}) \in \mathbf{m}^{(i)}$ to a subset of honest users $\mathcal{I} \subseteq \mathcal{H}$:

- **a)** Entry $(\mathcal{C}, \mathcal{P})$ in $\mathbf{m}^{(i)}$ is updated to $(\mathcal{C}, \mathcal{P}')$, where $\mathcal{P}' = \mathcal{P} \setminus \mathcal{I}$.
- **b)** Each $(\mathcal{C}, \mathcal{P}) \in \mathbf{m}^{(i)}$ must be delivered to all honest parties $\mathcal{H}$ by slot $\mathcal{C}.\mathsf{tip}.\mathsf{sl} + \delta$, and is then removed from $\mathbf{m}^{(i)}$.

Extend. To extend $\mathcal{C}$, party i generates $B = (t+1, H(\mathcal{C}.\mathsf{tip}), d, i)$ with an *ordering* of inputs $d = \{\mathsf{in}_i\}_{i\in[m]}$ input by environment $\mathcal{Z}$ for slot t.

Figure 1 Model of δ-PoS execution induced by environment $\mathcal{Z}$ and characteristic string w.

according to Figure 1, which precisely models the adversarial powers to influence the round-wise evolution of block tree structures in the local view parties as the full PoS protocol in [19], but omits details such as block proofs, signatures or individual leader election procedures such as evaluation of verifiable random functions. A given characteristic string w *induces* executions of our δ-PoS model that generate local tree structures identical to those resulting from a full PoS [19] protocol execution that *induces* a leader election sequence consistent with w and activates the same parties and adversarial actions.

Let the protocol execution state Γ_t in slot t, consist of honest party states, including the local block tree view $\mathcal{T}^{(i)}$ and the outbound message queue $\mathbf{m}^{(i)}$ for each honest party $i \in \mathcal{H}$. Further, let Γ_t include the blockchain tree view $\mathcal{T}^{\mathcal{A}}$ of the adversary.

$$\Gamma_t = \left(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}\}_{i\in\mathcal{H}}, \mathcal{T}^{\mathcal{A}}\right) \tag{2}$$

The outbound message queue $\mathbf{m}^{(i)} = \{(\mathcal{C}, \mathcal{P}), ...\}$ in Γ_t is the set of broadcast, yet undelivered chains previously sent by honest party i. For each entry $(\mathcal{C}, \mathcal{P}) \in \mathbf{m}^{(i)}$, $\mathcal{C}$ was initially broadcast and added to the local message queue at slot $\mathcal{C}.\mathsf{sl} \leq t$. Each entry in $\mathbf{m}^{(i)}$ consists of a chain $\mathcal{C}$ and honest party subset $\mathcal{P} \subset \mathcal{H}$, which has yet to receive the message. $\mathcal{A}$ is required to deliver all honestly broadcast chains with a delay of no more than δ slots. The model executes round-wise beginning from initial state Γ_0, where the tree views of all parties consist only of the genesis block.

In each round from slot t to $t+1$, the leader is implied by by $w_{t+1} \in \{0, 1, \perp\}$. For a uniquely honest slot, the environment $\mathcal{Z}$ is permitted to activate any honest party to extend the longest chain in its local view, where the inputs for insertion in the block are provided by $\mathcal{Z}$. We interpret $w_{\mathsf{slot}} = 1$ as a strictly adversarial slot, since the adversary could affect the structure of local trees views in the same way as multiple honest leaders: namely, by producing multiple blocks associated with the same slot.

PoS Security. The seminal work on formalizing the Bitcoin backbone protocol [23] proved *liveness* and *persistence* of longest-chain proof-of-work (PoW) protocols in terms of common-prefix, chain growth and chain quality properties, which are also achieved for PoS in Ouroboros Praos [19]. We restate these below and formally prove them for FairPoS in Section 5.

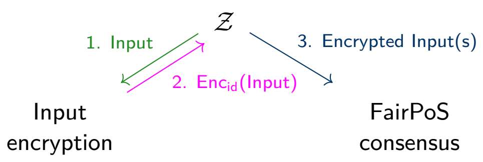
▶ **Definition 3** (**Common prefix, k-CP; with parameter $k \in \mathbb{N}$**). *The chains $\mathcal{C}_1, \mathcal{C}_2$ possessed by two honest parties at the onset of the slots $t_1 < t_2$ are such that $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$, where $\mathcal{C}_1^{\lceil k}$ denotes the chain $\mathcal{C}_1^{\lceil k}$ obtained by removing the last k blocks from $\mathcal{C}_1$, and $\preceq$ denotes the prefix relation.*

▶ **Definition 4** (**Chain growth, (τ, s)-CG; with parameter $\tau \in (0, 1]$ and $s \in \mathbb{N}$**). *Consider the chains $\mathcal{C}_1, \mathcal{C}_2$ possessed by two honest parties at the onset of two slots t_1, t_2 with t_2 at least s slots ahead of t_1. Then it holds that $\mathsf{len}(\mathcal{C}_2) - \mathsf{len}(\mathcal{C}_1) \geq \tau \cdot s$. We call τ the speed coefficient.*

▶ **Definition 5** (**Chain quality, (μ, k)-CQ; with parameters $\mu \in (0, 1]$ and $k \in \mathbb{N}$**). *Consider any portion of length at least k of the chain possessed by an honest party at the onset of a round; the ratio of blocks originating from the adversary is at most $1 - \mu$. We call μ the chain quality coefficient.*

4 The FairPoS protocol

As in δ-PoS, we model a FairPoS execution that is induced by an environment $\mathcal{Z}$ and a characteristic string w; both protocol participants and adaptive adversary $\mathcal{A}$ are spawned by $\mathcal{Z}$, whereas w governs which parties may be activated by $\mathcal{Z}$ to generate blocks at each slot. The adaptive adversary is permitted to spend its corruption budget on honest parties anytime; in particular, it can observe any message emitted by an honest party, and then decide its corruption strategy based on the message sent by the honest user. In FairPoS, inputs provided to the block leader by $\mathcal{Z}$ are delay encrypted. Thus, a party must first be activated by $\mathcal{Z}$ with a plain-text input and execute the input encryption procedure (Fig. 2). Upon receiving encrypted inputs, $\mathcal{Z}$ can forward an ordering of encrypted client inputs to the elected block leader in the FairPoS execution (Fig. 4).



We introduce the FairPoS model in parts. In Section 4.1, we formally define the notion of input fairness in the permissionless setting for clients sending transactions to a FairPoS execution. In order for an encrypted input to reach the k-common-prefix, the duration implied by the delay parameter d must be sufficiently long. We then define the

FairPoS input encryption

Let $\mathsf{KES} = (\mathsf{Gen},\mathsf{Sign},\mathsf{Verify},\mathsf{Update})$ denote a key evolving signature scheme and $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc},\mathsf{Dec})$ a symmetric-key encryption scheme. Let the genesis block of a chain contain a delay encryption parameter $\mathsf{DE.pk}$ and a chain tip imply account keys $\{\mathsf{KES.vk}_i\}_{i\in[n]}$.

Gen. Upon $(\textsc{Gen})$, set $(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{Gen}(1^k, T)$, return vk.

Sign. Upon $(\textsc{Sign}, \mathcal{C}, \mathsf{in})$

```
1. Let $\mathsf{id} = \mathcal{C}.\mathsf{tip}$, assert $\mathsf{vk} \in \mathsf{accts}(\mathcal{C}.\mathsf{tip})$
2. Set $\mathsf{pk}$ as $\mathsf{DE.pk}$ contained in chain genesis block $\mathcal{C}_0$.
3. Compute $(c, k) \leftarrow \mathsf{DE.Encaps}(\mathsf{pk}, \mathsf{id})$.
4. Encrypt input with key $k$: $m \leftarrow \mathsf{SKE.Enc}_k(\mathsf{in})$.
5. Generate $\sigma \leftarrow \mathsf{KES.Sign}_{\mathsf{sk}}(c \,|\, m \,|\, \mathsf{id})$.
6. Set $\mathsf{sk} \leftarrow \mathsf{Update}(\mathsf{KES.sk})$, erasing previous signing key.
7. Return $(c, m, \sigma)$.
```

Figure 2 Input encryption procedure in FairPoS.

FairPoS input encryption procedure; our protocol deploys key evolving signature schemes as in Ouroborous Praos [19], which prevents the adaptive adversary from obtaining static key material upon corrupting an honest party that has just emitted an honestly signed input.

In Sections 4.2 and 4.3, we introduce the formal FairPoS protocol execution model, which extends δ-PoS with key extraction and a novel longest-extendable-chain selection rule. Here, encrypted inputs are generated as in Section 4.1 and given by the environment $\mathcal{Z}$ to parties executing the protocol. We first describe how the adversary can prevent key extraction processes to complete on time by delaying adversarial blocks, thereby motivating the design of the longest-extendable-chain selection rule in the FairPoS model, which mitigates such adversarial impedance and ensures honest chain growth independent of chosen delay encryption parameter d.

A security analysis of FairPoS follows in Section 5, where the precise relationship between input fairness, chain growth, common-prefix and chain quality properties is formalized.

4.1 Input fairness & encryption

▶ **Definition 6 (Input fairness, IF).** *Consider the chain $\mathcal{C}$ possessed by an honest party at the onset of a round, where k-CP holds true. Input fairness holds if for all blocks $B \in \mathcal{C}^{\lceil k}$: 1. the adversary cannot decrypt an honestly encrypted input in B before B is in the k-common-prefix; 2. encrypted inputs in B are eventually decrypted by all honest parties.*

Input fairness is conditioned on k-common-prefix property in FairPoS. Intuitively, the extraction delay d in FairPoS must be parameterized, such that the encrypted input can reach the common-prefix before d time passes. For simplicity, we denote d as time in slots. Note that input fairness permits an encrypted input to *not* become finalized and decrypted by the adversary: we argue this outcome is acceptable as the client transaction is not executed and thus cannot be exploited in any input ordering attacks. This is consistent with [7, 32, 33].

We sketch the input encryption procedure for FairPoS shown in Figure 2, where the environment $\mathcal{Z}$ provides the plain-text input for a party to encrypt and sign. For a block B, inputs are encrypted to a session id which is set to the chain tip that B is extending, such that $\mathsf{id} = \mathcal{C}.\mathsf{tip}$ and $\mathcal{C}.\mathsf{tip} = B.\mathsf{st}$. To ensure that a slot leader cannot insert an encrypted input to a *later* block, potentially deferring its insertion until the key extraction is completed, we ensure that the input is *bound* to a child block of $\mathcal{C}.\mathsf{tip}$ with a signature.

In the adaptive corruption setting, we deploy an efficient key evolving signature scheme (KES) [8, 25] as in Ouroborous Praos [19]. Such schemes evolve secret key material *forward* with each signature, thereby erasing any information that could be used to generate verifying signatures of past rounds (See Appendix A of [16]). An adaptive adversary can always corrupt an honest user who has just *broadcast* a newly delay encrypted and signed input; with static key material only, the adversary would learn the signature key and generate verifying signatures of the delay encrypted input to insert it into any later block, potentially *after* decrypting the encrypted input[2]. In other words, since it is the signature of an honest user that binds its encrypted input to a specific block, allowing the adaptive adversary to bind the same honestly encrypted input to a later block will violate input fairness in Definition 6.

For the public verification of such signatures, we assume the presence of logical accounts for all parties, each associated with a public, signature verification key inferred from the chain tip.

Malformed inputs. As shown in Fig. 2, it will be possible for adversarial parties to encrypt malformed inputs; elected, honest block leaders which included these encrypted inputs to their blocks cannot discern the validity of the plaintext message, as the key extraction procedure is intended to proceed until the input reaches the k-common-prefix.

We omit a detailed treatment of mitigation techniques, but sketch several possible approaches. (1) A zero-knowledge proof of well-formedness may accompany each encrypted input, proving well-formedness of the underlying input. (2) Fees may mitigate denial-of-service attacks from mal-formed, adversarial inputs; we note that the correct execution- or gas-dependent fee amount can also be proven in zero-knowledge for certain applications.

We also highlight the existence of consensus protocols with explicit focus on input finalization for high transaction-throughput [4]. Here, finalized yet invalid inputs are simply ignored when determining canonical, total ordering of the chain.

4.2 Introducing key extraction in FairPoS

In FairPoS, each block is associated with a session id string, to which inputs in the child block are encrypted, as required by the input encryption procedure in Fig. 2. The extraction of the session key idk required to decrypt the finalized, input ciphers is parameterized to take d slots or rounds. A key design goal of FairPoS is to ensure that the session keys for each block will be included in *later blocks* of the same chain so that lightweight clients following the protocol execution do not need to perform expensive extractions to observe the blockchain state. This suggests that the protocol must define a fixed "extraction schedule" parameter $D \geq d$, such that the session key of a block B must be included the earliest block following slot $B.\mathsf{sl} + D$ of a chain extending B. Defining such a fixed extraction schedule, however, is not trivial, as the adversary can delay its own blocks arbitrarily, and reveal them to a selected subset of honest users only, and thereby induce additional block delays for other honest parties as the dishonest block must be relayed over the network, further hindering timely completion of key extraction processes run by honest parties.

[2] In practice, clients may be required to safeguard static key material. For modelling consistency, we assume clients performing the input encryption procedure can be adaptively corrupted just as parties participating in FairPoS consensus, motivating the use of key evolving signature schemes.

In this section, we provide preliminary definitions and intuition for how FairPoS achieves scheduled key extraction without breaking honest chain growth; the latter is formalized as the Δ-monotonicity property in Definition 7 which also holds in Praos [19]. Then, we define *receipt delays* in Definition 8; this is the delay between the onset of a round and the local arrival time of a block associated with the same round. Given receipt delays, we then show a naive application of maximum receipt delay limits for blocks. The idea here is to ensure key extraction processes are initiated in a timely manner by requiring blocks to arrive within a maximum duration after their scheduled generation. We then show how the adversary can attack such a naive scheme and cause chain stalls (eq. 3). This attack is overcome in FairPoS by carefully incrementing maximum receipt delays for blocks further away from the chain-tip, as illustrated in Example 9. We demonstrate that FairPoS achieves Δ-monotonicity in Theorem 10, a key property required for FairPoS security (§5). A more formal treatment of the full FairPoS consensus protocol follows in Section 4.3.

▶ **Definition 7** (**Δ-Monotonicity**, from [19]). *Let $\mathcal{T} = \bigcup_{i \in \mathcal{H}} \mathcal{T}^{(i)}$ be an honest tree rooted in genesis resulting from an execution of protocol π in a δ-synchronous network: it consists of all chains broadcast by honest parties. Further, let $\mathsf{depth}^{\mathcal{T}}(i)$ denote the length of the chain in $\mathcal{T}$ extended by the uniquely honest leader of slot i. $\mathcal{T}$ exhibits the Δ-monotonicity property if the following holds true.*

$$\text{For all uniquely honest slots } (i,j) \text{ s.t. } j \geq i + \Delta: \quad \mathsf{depth}^{\mathcal{T}}(j) > \mathsf{depth}^{\mathcal{T}}(i)$$

In PoS executed in a δ-synchronous setting, δ-monotonicity is trivially achieved: any honest block tip must arrive in the view of other honest party after δ slots, and is thus considered as a chain candidate for extension by any honest leader applying the longest chain selection rule.

We introduce a formal notion of *receipt delay*, which, informally, quantifies how far a local key extraction process is "behind schedule" due to adversarial delay of block arrival.

▶ **Definition 8** (**Receipt delay**). *Let $\mathsf{r}^{(i)}(B) : \mathcal{B} \rightarrow \mathbb{Z}_0$ be the delay in slots between $B.\mathsf{sl}$ and the local arrival of block B from the view of the party (i). If B is an empty-block $(\perp)$, we define the receipt delay to be $\perp$.*

When the extraction window is defined as a slot interval preceding genesis, where slot indices are "negative", let the receipt delay is defined as $\perp$. We allow a receipt delay of $\perp$ to be interpreted as a receipt delay of 0.

Attack on receipt delay consistency. The receipt delay of each block B must be bounded by the protocol in order to guarantee a fixed future slot in which the honest party can complete the extraction of the session key of B and produce a valid block. However, note that the receipt delay is defined on the local view of a single party; the adversary can trivially achieve inconsistencies in receipt delays between honest parties, such that arrival of a block is considered "on-time" by party, but "too late" by another. We illustrate such an attack on receipt delay consistency (Eq. 3) between honest parties.

$$\mathcal{A} \xrightarrow{\mathcal{C}_{\mathcal{A}}} \mathcal{P}_i \xrightarrow{\mathcal{C}_{\mathcal{A}}|B_i} \{\mathcal{P}_j\}_{j \in \mathcal{H}} \tag{3}$$

For simplicity, consider a chain $\mathcal{C}_{\mathcal{A}}$, which consists of adversarial blocks only. Then, let the adversary forward the chain to an honest party $\mathcal{P}_i$, such that all receipt delays for each block in $\mathcal{C}_{\mathcal{A}}$ are exactly the maximum receipt delay permitted by the protocol. Should honest $\mathcal{P}_i$

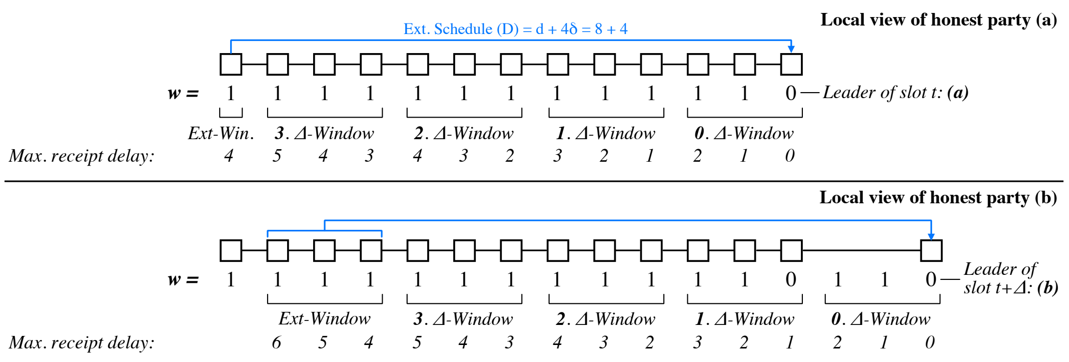

Local view of honest party (a)
Ext. Schedule (D) = d + 4δ = 8 + 4
w = 1 1 1 1 1 1 1 1 1 1 1 1 0
Leader of slot t: (a)
Ext-Win.
3. Δ-Window
2. Δ-Window
1. Δ-Window
0. Δ-Window
Max. receipt delay: 4 5 4 3 4 3 2 3 2 1 2 1 0
Local view of honest party (b)
w = 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0
Leader of slot t+Δ: (b)
Ext-Window
3. Δ-Window
2. Δ-Window
1. Δ-Window
0. Δ-Window
Max. receipt delay: 6 5 4 5 4 3 4 3 2 3 2 1 2 1 0

Figure 3 Key extraction in (d=8, δ=1, Δ=3)-FairPoS with maximally permitted receipt delays.

extend $\mathcal{C}_\mathcal{A}$ and subsequently deliver $\mathcal{C}_\mathcal{A} \mid B_i$ to all other honest parties $\{\mathcal{P}_j\}_{j\in\mathcal{H}:j\neq i}$ with a network delay of δ slots, the receipt delays of blocks in $\mathcal{C}_\mathcal{A}$ in the view of the other honest parties will violate the protocol, as these will be $\mathbf{r}^{(j)}(B) = \mathbf{r}^{(i)}(B) + \delta$ for each $B \in \mathcal{C}_\mathcal{A}$; by assumption, $\mathbf{r}^{(j)}(B)$ is the maximum permitted by the protocol. Thus, the honest chain tip B_i will never be extended by other honest parties, as its prefix $\mathcal{C}_\mathcal{A}$ contains blocks with invalid receipt delays, violating Δ-monotonicity for any Δ; in the view of other parties, $\mathcal{C}_\mathcal{A} \mid B_i$ is an invalid chain.

FairPoS key extraction. We overcome this class of attacks in FairPoS by permitting increasing receipt delay limits for blocks that are farther away from the chain tip (Figure 4). The high-level idea is as follows: To achieve Δ-monotonicity (Def. 7), we consider honest block leaders of slots which are Δ slots apart. Even if the receipt delay view of a block may differ from one honest leader at slot t to the leader of a later slot $t+\Delta$, the leader at $t+\Delta$ will grant the same block a higher, maximum receipt delay to account for any additional network delays induced by the adversary, as shown above. Our protocol permits independent choices of extraction delay (d), network delay (δ), montonicity parameter (Δ) satisfying Eq. 5. We provide a discussion of parameterizations of (d, δ, Δ)-FairPoS in Sec. 4.3. Before formalising the FairPoS key extraction protocol, we provide an example parameterization of FairPoS for intuition.

▶ **Example 9.** We illustrate an execution of (d=8, δ=1, Δ=3)-FairPoS in Fig. 3, where the local chain view of honest parties (a) and (b) is shown. Here, party (a) is elected block leader at slot t, whereas party (b) is elected block leader at slot $t+\Delta$; for Δ-monotonicity to hold, the chain extended by (a) must be extendable by (b); in party (b)'s local view, this requires all blocks in the chain to respect the receipt delays imposed by the protocol, even if the adversary has induced *inconsistent receipt delays* between honest parties. For simplicity, we assume no leaderless slots ($\perp$) in this example.

Receipt delay limits ("max. receipt delay" in Fig. 3) imposed by FairPoS are organised in slot intervals, called **Δ-windows**. Beginning from the current slot, preceding slots are divided into slot windows of $\Delta = 3$ length. Each slot position within a n'th Δ-window is granted an *additional* receipt delay of delta in the subsequent $(n+1)$'th Delta window. The **extraction window** defines the blocks for which the session keys must be extracted and included into the block of the current slot; note that the **extraction schedule** ($D = d + 4\delta = 12$) is consistent with the maximum receipt delay ($4\delta = 4$) imposed on blocks in the extraction window.

Fig. 3 illustrates a worst case scenario; honest party (a) is extending a fully adversarial chain, where the adversary has forwarded the blocks to party (a) only, with the maximally permitted receipt delays. Honest party (a) broadcasts the chain upon extending it in slot t, such that all blocks arrive at party (b) with an additional, worst case network delay $\delta = 1$. However, note that in the view of block leader party (b), each block of the chain in the n'th Δ-window is now in the $(n+1)$'th Δ-window, which tolerates an additional receipt delay of $\delta = 1$. Thus, this chain from party (a) is extendable by party (b) after Δ slots, implying Δ-monotonicity for $\Delta = 3$.

We highlight that each honest party has no knowledge of whether blocks were generated by an adversary or honest party. Further, no honest party can infer the true receipt delays in the local views of other honest parties. Informally, the FairPoS key extraction protocol is designed to accommodate the "worst case" adversarial interference shown in this example.

4.3 The (d, δ, Δ)-FairPoS consensus protocol

We formalize the key extraction protocol introduced in §4.2 and present the full FairPoS execution model (Fig. 4) which also extends δ-PoS (Fig. 1) with a novel longest-extendable-chain selection rule. The design of FairPoS permits scheduled key extraction for each block whilst maintaining Δ-monotonicity (Def. 10), essential for the security of FairPoS (§5).

Extraction window. A FairPoS protocol instance is parameterized with extraction schedule D, the duration after which the extracted session keys of a block are due in the next, available block. Here, we must account for the possibility of a *gap* between the chain tip and the current slot, for which no extracted keys could have been inserted. We therefore require a notion of an extraction window that denotes a window of preceding slots, for which associated blocks have key extractions due in the current slot. We first formalize $\mathsf{gap2tip}(t, \mathcal{C})$ as the number of *empty* slots between slot t and $\mathcal{C}.\mathsf{tip}.\mathsf{sl}$.

$$\mathsf{gap2tip}(t, \mathcal{C}) = t - \mathsf{tip}(\mathcal{C}).\mathsf{sl} \quad \text{for} \quad t > \mathsf{tip}(\mathcal{C}).\mathsf{sl}$$

Then, the extraction window of current slot t and chain $\mathcal{C}$ with extraction schedule D can be defined as the range of slots, which are at least D slots in the past and are associated with blocks in $\mathcal{C}$ *without* key extractions already inserted in $\mathcal{C}$ due to a non-empty $\mathsf{gap2tip}(t, \mathcal{C})$.

$$\mathsf{extWin}_D(t, \mathcal{C}) = (t - D - \mathsf{gap2tip}(t, \mathcal{C}) : t - D] \tag{4}$$

Extraction schedule & Δ-windows. Let (d, δ, Δ)-FairPoS be parameterized by delay encryption parameter $d > 0$, maximum network delay $\delta \geq 0$ and desired monotonicity parameter $\Delta > \delta$ (Definition 7). Parameters d, δ, and Δ are chosen to satisfy the following relation. Note that n denotes the number of Δ-windows in extraction schedule D (see Δ-windows in Fig. 3).

$$D = n\Delta \quad \text{where} \quad n = d/(\Delta - \delta) \text{ s.t. } n \in \mathbb{Z} \tag{5}$$

In words: the extraction schedule D is divisible by Δ. Furthermore, the number of Δ-windows in D must equal $d/(\Delta - \delta)$. We define the m'th "Δ-window" of the current slot t as the following slot interval, consistent with Figure 3.

$$\Delta\mathsf{Win}_D(t, m) = (t - (m+1)\Delta : t - m\Delta] \quad \text{for} \quad m \in [0 : \frac{D}{\Delta}) \tag{6}$$

Chain extendability. A chain $\mathcal{C}$ is extendable by leader of slot t with local receipt delay view $\mathbf{r}^{(i)}$ if the receipt delays specific to each slot and Δ-window are satisfied, as implied by conditions ①, ② and ③ in Equation (7).

$$\mathsf{ext}(t, \mathcal{C}, \mathbf{r}^{(i)}) = \begin{cases} 1 & ① \wedge ② \wedge ③ \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{array}{lll} ① & \forall m \in [0 : \frac{D}{\Delta}) : & \\ & \forall j \in \Delta\mathsf{Win}_D(t, m) : & \mathbf{r}^{(i)}(\mathcal{C}_j) \leq m\delta + (t - m\Delta - j) \\ ② & \forall j \in \mathsf{extWin}_D(t, \mathcal{C}) : & \mathbf{r}^{(i)}(\mathcal{C}_j) \leq \frac{D}{\Delta}\delta + (t - D - j) \\ ③ & \forall B \in \mathcal{C} : & B \text{ contains valid } \mathsf{idk} \text{ for each block in its } \mathsf{extWin} \end{array} \tag{7}$$

Concretely, condition ① reflects the maximum permitted receipt delays for each slot in the m'th Δ-window. Condition ② describes the maximum permitted receipt delays for slots in the extraction window (Eq. 4). Observe that these conditions are satisfied in the view of parties (a) and (b) in Figure 3 since the maximum receipt delays imposed by FairPoS are satisfied. Condition ③ states that all blocks in the chain must include valid session keys from past, parent blocks in their respective extraction windows.

▶ **Theorem 10.** *(Δ-Monotonicity of FairPoS) Every protocol execution of (d, δ, Δ)-FairPoS results in an honest tree $\mathcal{T}$ that exhibits the Δ-monotonicity property.*

Proof (Sketch). Δ-monotonicity holds, because each slot in the m'th Δ-window is granted an additional δ receipt delay budget in the $m+1$'th Δ-window. Thus, if a chain is extendable (eq. 7) in the view of an honest party, the same chain must be extendable by all other parties following δ slots, despite the attack shown in eq. 3 that induces inconsistent receipt delays. Intuitively, the protocol can tolerate worst-case, receipt delay inconsistencies of δ slots from such attacks because each block is granted an additional δ in delay budget after Δ slots. The formal proof of Theorem 10 is stated in Appendix C of [16]. ◀

FairPoS execution model. We can now state the full FairPoS protocol and its execution model in Fig. 4. FairPoS extends the δ-PoS model in Figure 1 with the longest-extendable-chain selection rule, key extractions and the insertion of delay encrypted inputs, generated by the procedure in Figure 2. The protocol execution state of (d, δ, Δ)-FairPoS is extended with local receipt delays:

$$\Gamma_t = \left(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}, \mathbf{r}^{(i)}\}_{i \in \mathcal{H}}, \mathcal{T}^{\mathcal{A}}\right) \tag{8}$$

Views in initial state Γ_0 contain a genesis block, which includes public parameters $\mathsf{DE.pk}, \mathsf{DE.ek}$. Importantly, we *require* the adversary $\mathcal{A}$ and each honest party to perform exactly one extraction step for each pending key extraction in each round of the execution. Thus, no party or adversary gains a time advantage in extracting session keys from blocks (See E1 in Figure 4). It remains to formally define the longest extendable chain selection in the local view of an honest party.

Longest-extendable-chain selection. In FairPoS, an honest slot leader chooses to extend the *longest extendable chain* in its local tree view $\mathcal{T}^{(i)}$, where chain extendability was previously defined in Eq. 7.

$$\mathsf{maxExtChain}(t, \mathcal{T}^{(i)}, \mathbf{r}^{(i)}) = \underset{\mathcal{C} \in \mathcal{T}^{(i)} : \mathsf{ext}(t, \mathcal{C}, \mathbf{r}^{(i)})}{\arg\max} \; \mathsf{len}(\mathcal{C}) \tag{9}$$

(d, δ, Δ)-FairPoS consensus model

The (d, δ, Δ)-FairPoS state $\Gamma_t = \left(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}, \mathbf{r}^{(i)}\}_{i\in\mathcal{H}}, \mathcal{T}^{\mathcal{A}}\right)$ evolves in a single round induced by environment $\mathcal{Z}$, adversary $\mathcal{A}$ and characteristic string w as follows.

Trees. Honest $\{\mathcal{T}^{(i)}\}_{i\in\mathcal{H}}$ and adversarial $\mathcal{T}^{\mathcal{A}}$ evolve as in δ-PoS (Fig. 1) except that the honest leader i extends the **longest-extendable chain** (Eq. 9) in its view $\mathcal{T}^{(i)}$.

Msgs. Honest message queues $\{\mathbf{m}^{(i)}\}_{i\in\mathcal{H}}$ evolve as in δ-PoS (Fig. 1).

Receipt delays. The receipt delays $\{\mathbf{r}^{(i)}\}_{i\in\mathcal{H}}$ in Γ_t evolve as follows:

R1 For each chain $\mathcal{C}$ delivered to honest party i by $\mathcal{A}$ at slot $t+1$: for each newly seen block $\mathcal{C}_j \in \mathcal{C}$, party i records $\mathbf{r}^{(i)}(\mathcal{C}_j) \leftarrow t + 1 - \mathcal{C}_j.\mathsf{sl}$.

Extraction. Each honest party $i \in \mathcal{H}$ and adversary $\mathcal{A}$ performs:

E1 In each round, exactly a single Extract step on each block in its local view for which key extraction is pending.

Extend. Party i generates $B = (t, H(\mathcal{C}.\mathsf{tip}), d_{\mathsf{ext}}|d_{\mathsf{ins}}, P_i)$ where

- $d_{\mathsf{ext}} = (\mathsf{idk}, \mathsf{idk}', \ldots)$ are extractions of blocks in $\mathcal{C}$ within $\mathsf{extWin}_D(t, \mathcal{C})$ (Eq. 4).
- $d_{\mathsf{ins}} = \{(c_j, m_j, \sigma_j)\}_{j\in[n]}$ is an ordering of encrypted inputs from enviroment $\mathcal{Z}$; Upon receiving d_{ins} from $\mathcal{Z}$, party i asserts for each entry; $\exists \mathsf{vk}_j \in \mathsf{accts}(\mathcal{C}.\mathsf{tip}) : 1 \leftarrow \mathsf{KES.Verify}_{\mathsf{vk}_j}((c_j|m_j|\mathcal{C}.\mathsf{tip}), \sigma_j)$.

Then, party i adds chain $\mathcal{C}' = \mathcal{C} \mid B$ to its local view.

Figure 4 Model of FairPoS execution induced by environment $\mathcal{Z}$ and characteristic string w. Encrypted inputs are generated in the input procedure in Fig 2 of §4.1.

4.4 Parameterization of FairPoS

We note that in parameterizing (d, δ, Δ)-FairPoS, the maximum network delay δ is conservatively chosen to reflect network realities. For given δ, monotonicity parameter Δ and key extraction delay d can be chosen quasi-independently. For given choices of $\Delta > \delta$, any d is permitted that satisifies Equation (5); namely, d must be any multiple of $(\Delta - \delta)$. In Section 5, we show that the probability of input fairness being violated by the adversary falls exponentially in d (Theorem 19). Thus, the security of input fairness can be chosen independently of that of common-prefix, chain growth and chain quality properties. This is important for utility; a larger d increases the robustness of input fairness, but this comes at a cost of the freshness of blockchain state.

Freshness of visible blockchain state. All proposed encrypt-and-reveal approaches (Section 2) to input fairness naturally accept a compromise in the "freshness" of visible blockchain state; a delay parameter d naturally induces a gap between the current slot, and the most recent block in the past with already decrypted inputs. In practice, clients must submit inputs which are then applied to a blockchain state which is not yet decrypted - in other words, clients must generate inputs based on older blockchain state. In applications with frequent inputs accessing the same public state, such as in decentralized finance [36], this likely results in increased number of invalid or reverted transactions, affecting the utility of the blockchain. In decentralized exchanges, for example, the market price is constantly

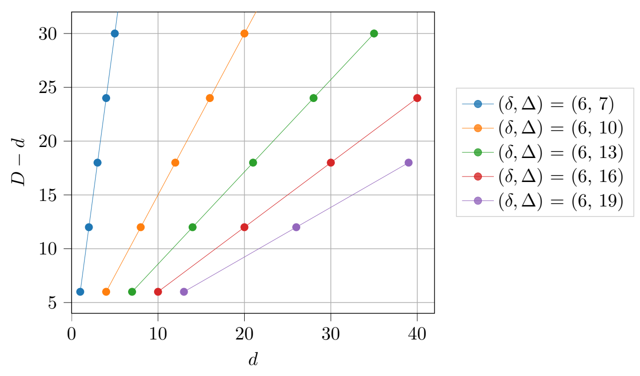


Figure 5 The *decryption gap* $(D - d)$ between key extraction schedule (D) and key extraction delay (d) is shown for selected parameterizations of (d, δ, Δ)-FairPoS. Decryption gap $D - d$ can be interpreted as the "efficacy" of the key extraction protocol in FairPoS, which aims to provide extracted keys to blocks as soon as possible.

adjusted with each submitted trade. A trade order with a price limit informed by an outdated price can become invalidated, as the older, lower price is no longer be available when the encrypted trade order is finally decrypted. We emphasize that the trade-off between utility and robustness of input fairness (Thm. 19) in FairPoS can be adjusted independently of the structural blocktree properties of common-prefix (Thm. 15), chain growth (Thm. 16) and chain quality (Thm. 17); in some applications a small, yet non-negligible probability of front-running may be acceptable for a fresher, decrypted blockchain state offered by a smaller delay extraction d parameter.

Decryption gap. Parameterizations of (d, δ, Δ)-FairPoS always imply an extraction schedule greater than the extraction delay parameter $(D > d)$ for non-zero network delay parameter δ (Eq. 5). Let $D - d$ denote the "decryption gap"; informally, it represents how much sooner the consensus node performing key extraction can decrypt a block compared to a light-weight client, which must wait for the extracted key to be included in a later block. This "slack" in extraction schedule D is necessary to maintain consensus security due to adversarial block delays attacking receipt delay consistency between honest users illustrated in §4.2. We emphasize that the decryption gap $D - d$ does not represent a compromise in input fairness or structural security properties; instead, it reflects the efficacy of the FairPoS key extraction protocol, which aims to provide extracted keys to lightweight clients as soon as possible after these are extracted.

Figure 5 shows the "decryption gap" $D - d$ for selected protocol parameters $(d, \delta = 6, \Delta)$. We highlight the first trade-off between $\Delta - \delta$ and decryption gap $D - d$; a higher monotonicity parameter Δ in $\Delta - \delta$ implies slower chain extendability, but can be exchanged for a smaller decryption gap, implying higher chain utility for light clients. Furthermore, delay encryption delay parameter d and decryption gap $D - d$ are linearly correlated for given δ and Δ; designing a secure key extraction protocol with sublinear decryption gap represents an interesting problem for future work.

5 FairPoS security

Having presented a formal model of (d, δ, Δ)-FairPoS we proceed to demonstrate that it satisfies common-prefix (Def. 15), chain growth (Thm. 16), chain quality (Thm. 17) and input fairness (Thm. 19).

Informally, common-prefix is demonstrated by showing that any execution of (d, δ, Δ)-FairPoS produces local blocktree views which could have resulted from a Δ-PoS execution; since common-prefix is interpreted as a structural branching property of local blocktree views, this allows us to directly inherit common-prefix from Δ-PoS previously proven in Praos [19]. Chain growth and chain quality are implied by Δ-monotonicity of FairPoS, previously shown in Theorem 10. Finally, input fairness in FairPoS is inferred from common-prefix, chain growth and chain quality.

5.1 Common-prefix in FairPoS

Demonstrating k-common-prefix in FairPoS is accomplished by formally relating the tree views generated in an execution of (d, δ, Δ)-FairPoS with those that could have resulted from δ-PoS. Let the *honest tree* of protocol execution state

$$\Gamma_t = \big(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}, \mathbf{r}^{(i)}\}_{i\in\mathcal{H}}, \mathcal{T}^{\mathcal{A}}\big)$$

be the union of honest tree views at slot t:

$$\mathcal{T}^{\mathcal{H}}(\Gamma_t) = \bigcup_{i\in\mathcal{H}} \mathcal{T}^{(i)} \tag{10}$$

In the analysis of PoS [19, 30], the branching structure of the honest tree informs us about events where a local chain was abandoned for a longer, alternative branch according to the *longest chain* selection rule. Informally, the common-prefix property is violated if the k-common-prefix shared between prior and newly adopted chains if their shared prefix is "too short". We begin our analysis with the (viable) branches which could have been adopted by honest parties in the first place.

Viable branches in PoS. A viable branch $\mathcal{B}$ in a tree $\mathcal{T}$ must exceed all honest chains-tips generated more than δ slots prior to $\mathcal{B}$.tip.slot: this property arises from the honest application of the *longest chain rule*. The honest leader of $\mathcal{B}$.tip.slot must have received any honest chain-tips generated δ slots prior and considered them as *alternative candidates* for extension. Therefore, the *viable* branch $\mathcal{B}$ must exceed these in length. For branch $\mathcal{B} \in \mathcal{T}$, we first formalize its set of alternative chain candidates as follows.

$$\begin{gathered}\mathbf{altChns}_\delta(\mathcal{B}, \mathcal{T}) = \\ \{\, \mathcal{C} \subseteq \mathcal{T} \mid \mathcal{C}.\mathsf{tip.ldr} \in \mathcal{H} \;\wedge\; \mathcal{B}.\mathsf{tip.sl} > \mathcal{C}.\mathsf{tip.sl} + \delta \,\}\end{gathered} \tag{11}$$

For a PoS protocol execution state $\Gamma_t = \big(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}\}_{i\in\mathcal{H}}, \mathcal{T}^{\mathcal{A}}\big)$, the set of well-formed, viable branches is formalized as the set of branches with lengths exceeding their honest, alternative chains.

$$\begin{gathered}\mathbf{viableBranches}^{\mathsf{PoS}}_\delta(\Gamma_t) = \\ \{\forall \mathcal{B} \in \mathcal{T}^{\mathcal{H}}(\Gamma_t) \mid \mathsf{len}(\mathcal{B}) > \underset{\mathcal{C}\,\in\,\mathbf{altChns}_\delta(\mathcal{B}, \mathcal{T}^{\mathcal{H}}(\Gamma_t))}{\arg\max} \mathsf{len}(\mathcal{C}) \;\}\end{gathered} \tag{12}$$

Viable chains in FairPoS. The notion of viable branches must be strengthened for FairPoS since the *longest-extendable-chain* rule introduces additional constraints for the adoption of a chain in the local honest tree view. Let the *extendable prefix* of a branch $\mathcal{B}$ in the view of honest parties at slot t be defined as the "longest extendable prefix" of a branch.

$$\mathbf{extPrefix}(t, \mathcal{B}, \{\mathbf{r}^{(i)}\}_{i\in\mathcal{H}}) = \underset{\mathcal{C}\preceq\mathcal{B}\,:\,\exists i\in\mathcal{H}\,:\,\mathsf{ext}(t,\mathcal{C},\mathbf{r}^{(i)})}{\arg\max} \mathsf{len}(\mathcal{C}) \tag{13}$$

For a (d, δ, Δ)-FairPoS state, let the set of viable chains be defined as the extendable prefixes (Equation (13)) of branches in the honest tree with lengths which exceed those of its alternative chains (Equation (11)) generated Δ slots prior: by the Δ-monotonicity property (Theorem 10), these chains must have been extendable by the leader that generated the respective prefix and considered these as candidates for extension. Viable chains in FairPoS are defined as;

$$\begin{aligned}&\mathbf{viableChains}_{\Delta}^{\mathsf{FairPoS}}(\Gamma_t) = \{\, \mathcal{C} \subseteq \mathcal{T}^{\mathcal{H}}(\Gamma_t) \mid ① \wedge ② \,\}\\ &① \ \exists \mathcal{B} \in \mathcal{T}^{\mathcal{H}}(\Gamma_t) : \mathcal{C} = \mathbf{extPrefix}(t, \mathcal{B}, \{\mathbf{r}^{(i)}\}_{i \in \mathcal{H}})\\ &② \ \mathsf{len}(\mathcal{C}) > \underset{\mathcal{C}' \in \mathbf{altChns}_{\Delta}(\mathcal{C}, \mathcal{T}^{\mathcal{H}}(\Gamma_t))}{\arg\max} \mathsf{len}(\mathcal{C}')\end{aligned} \tag{14}$$

In words; a viable chain in FairPoS and the local honest view must ① be an extendable prefix and ② this prefix must also exceed its alternative chains in length.

Next, we restate the *divergence* notion from Praos [19, 30] which formally describes the *magnitude* of branching caused by the switching between viable chains.

▶ **Definition 11 (Divergence).** *For two chains $\mathcal{C}_1$ and $\mathcal{C}_2$, define their divergence to be the quantity*

$$\mathsf{div}(\mathcal{C}_1, \mathcal{C}_2) = \mathsf{min}(\mathsf{len}(\mathcal{C}_1), \mathsf{len}(\mathcal{C}_2)) - \mathsf{len}(\mathcal{C}_1 \cap \mathcal{C}_2)$$

where $\mathcal{C}_1 \cap \mathcal{C}_2$ denotes the common prefix of $\mathcal{C}_1$ and $\mathcal{C}_2$. We extend this notion of divergence to the protocol execution state Γ resulting from the execution of protocol π induced by characteristic w in the δ-synchronous setting: here, the maximum divergence over any two viable chains is quantified.

$$\mathsf{div}_{\delta}^{\pi}(\Gamma) = \mathsf{max}_{\mathcal{C}_1, \mathcal{C}_2 \in \mathsf{viableBranches}_{\delta}^{\pi}(\Gamma)} \mathsf{div}(\mathcal{C}_1, \mathcal{C}_2)$$

Finally, we define the divergence of a characteristic string w to be the maximum divergence observable over all states which could have resulting from protocol executions induced by w. More formally, let $\mathsf{exec}_{\delta}^{\pi}(\Gamma_0, w)$ denote all possible executions of π beginning with state Γ_0 which could have been induced by w in a δ-synchronous network. Then the divergence of a characteristic string w is defined as:

$$\begin{aligned}\mathsf{div}_{\delta}^{\pi}(w) &= \mathsf{max}_{\Gamma \in \mathsf{reachable}_{\delta}^{\pi}(\Gamma_0, w)} \mathsf{div}_{\delta}^{\pi}(\Gamma)\\ \textit{where } \mathsf{reachable}_{\delta}^{\pi}(\Gamma_0, w) &= \{\Gamma \mid \exists \lambda \in \mathsf{exec}_{\delta}^{\pi}(\Gamma_0, w) : \Gamma_0 \xrightarrow{\lambda} \Gamma\}\end{aligned} \tag{15}$$

For PoS, the probability that the divergence exceeds k-blocks over an execution of R slots, is given by the following theorem from [19]. Bounding the probability of divergence naturally implies common-prefix security.

▶ **Theorem 12.** *(PoS Divergence, Theorem 4 in [19]) Let active slot coefficient $f \in (0, 1]$ and α be such that $\alpha(1 - f)\Delta = (1 + \epsilon)/2$ for some $\epsilon > 0$. Further, let w be a string drawn from $\{0, 1, \perp\}^R$ according to D_{α}^{f}. Then we have $\Pr_{w \leftarrow \$ D_{\alpha}^{f}}[\mathit{div}_{\Delta}^{\mathit{PoS}}(w) \geq k] \leq \exp(\ln(R) - \Omega(k - \Delta))$.*

Towards demonstrating k-common prefix in FairPoS, we first present a central theorem, which states that for all executions of (d, δ, Δ)-FairPoS in the δ-synchronous setting, there exists an execution of PoS in the Δ-synchronous setting, such that viable chains of the d, δ, Δ-FairPoS honest tree are *equivalent* ($\equiv$) to the viable branches of the PoS honest tree: here, we define the equivalence of chains such that only their structural properties are considered, formally stated in Appendix B of [16].

▶ **Theorem 13 (Equivalent trees).** *For any (d, δ, Δ)-FairPoS execution λ induced by a characteristic string $w \in \{0, 1, \perp\}^*$, $\Gamma_0 \to^{\lambda} \Gamma$, there exists a Δ-PoS execution λ' induced by the same w, $\Gamma'_0 \to^{\lambda'} \Gamma'$, such that the viable chains in Γ are equivalent to the viable branches in Γ'.*

$$\begin{aligned} \forall w \in \{0,1,\perp\}^* : \quad & \forall \lambda \in \mathit{exec}_{\delta}^{\mathit{FairPoS}}(\Gamma_0, w), \Gamma_0 \xrightarrow{\lambda} \Gamma : \\ & \exists \lambda' \in \mathit{exec}_{\Delta}^{\mathit{PoS}}(\Gamma'_0, w), \Gamma_0 \xrightarrow{\lambda'} \Gamma' : \\ \boldsymbol{viableChains}_{\delta}^{\mathit{FairPoS}}(\Gamma) \equiv \boldsymbol{viableBranches}_{\Delta}^{\mathit{PoS}}(\Gamma') & \end{aligned}$$

We refer to the full proof of Theorem 13 in Appendix C of [16]; here, we demonstrate how to match any honest and adversarial action in Δ-PoS with a corresponding honest or adversarial action in (d, δ, Δ)-FairPoS such that the viable branches of (d, δ, Δ)-FairPoS evolve in lockstep with the viable chains in the Δ-PoS execution.

Since divergence is defined over viable chains and viable branches, we can infer Corollary 14 from Theorem 13.

▶ **Corollary 14.** $\forall w \in \{0, 1, \perp\}^* : \mathit{div}_{\delta}^{\mathit{FairPoS}}(w) \leq \mathit{div}_{\Delta}^{\mathit{PoS}}(w)$

This allows us to infer k-common-prefix from bounding the probability of the event $\mathsf{div}_{\Delta}^{\mathsf{PoS}}(w) > k$ for $w \leftarrow\$ \mathcal{D}_{\alpha}^{f}$ in PoS [19].

▶ **Theorem 15 (k-Common prefix in FairPoS).** *Let $\mathcal{A}$ be an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1 - \alpha)$ stake, where α be such that $\alpha(1 - f)\Delta = (1 + \epsilon)/2$ for active slot coefficient $f \in (0, 1]$ and some $\epsilon > 0$. The probability that $\mathcal{A}$ makes the protocol violate the k-common-prefix property in a δ-synchronous environment throughout a period of R slots is no more than $\exp(\ln R + \Delta - \Omega(k))$.*

Proof. Let w be drawn from dominant distribution $\mathcal{D}_{\alpha}^{f}$ (Equation (1)), with honest stake α and parameter f satisfying $\alpha(1 - f)\Delta = (1 + \epsilon)/2$ for some $\epsilon > 0$. From Corollary 14 and Theorem 12 we infer the following:

$$\Pr_{w \leftarrow \mathcal{D}_{\alpha}^{f}}[\mathsf{div}_{\delta}^{\mathsf{FairPoS}}(w) \geq k] \leq \Pr_{w \leftarrow \mathcal{D}_{\alpha}^{f}}[\mathsf{div}_{\Delta}^{\mathsf{PoS}}(w) \geq k] \leq \exp(\ln R + \Delta - \Omega(k)) \tag{16}$$

From Corollary 14, $\mathsf{div}_{\delta}^{\mathsf{FairPoS}}(w) \geq k \implies \mathsf{div}_{\Delta}^{\mathsf{PoS}}(w) \geq k$, implying the left equality in Equation (16). The right inequality is inferred from Theorem 12. ◀

5.2 Chain growth, chain quality and input fairness

Both chain growth and chain quality of FairPoS can be derived from Δ-monotonicity of FairPoS (Theorem 10) and probabilities bounding security failure from PoS [19]. We refer to Appendix C in [16] for the full proofs of the following security theorems.

▶ **Theorem 16 ((τ, s)-Chain growth in (d, δ, Δ)-FairPoS).** *Let $\mathcal{A}$ be an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1 - \alpha)$ stake, where α be such that $\alpha(1-f)\Delta = (1+\epsilon)/2$ for active slot coefficient $f \in (0, 1]$ and some $\epsilon > 0$. Then the probability that $\mathcal{A}$ makes the protocol violate the chain growth property with parameters $s \geq 4\Delta$ and $\tau = c\alpha/4$ throughout a period of R slots, is no more than $\exp(-c\alpha s/(20\Delta) + lnR\Delta + O(1))$, where c denotes the constant $c := c(f, \Delta) = f(1 - f)^{\Delta}$.*

▶ **Theorem 17 ((μ, k)-Chain quality in (d, δ, Δ)-FairPoS).** *Let $\mathcal{A}$ be an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1 - \alpha)$ stake, where α be such that $\alpha(1 - f)\Delta = (1 + \epsilon)/2$ for active slot coefficient $f \in (0, 1]$ and some $\epsilon > 0$. Then the probability that $\mathcal{A}$ makes FairPoS violate the chain quality property with parameters k and $\mu = 1/k$ throughout a period of R slots, is no more than* $\exp(\ln R - \Omega(k))$.

Input fairness is obtained from chain growth, common prefix and chain quality. Informally, given time d and chain growth rate τ, we can determine common-prefix and chain quality parameters such that a decrypted input must have sufficient time (d slots) to reach finalization or lie in an abandoned chain.

▶ **Lemma 18 (Input fairness from CG, CP and CQ in (d, δ, Δ)-FairPoS).** *Input fairness is implied in an execution of (d, δ, Δ)-FairPoS, in which (τ, d)-chain growth, $(d\tau(\tau - \delta/(\Delta - \delta)) - 1)$-common prefix and $(1/(D + 1), D + 1)$-chain quality hold, where $D = d\Delta/(\Delta - \delta)$*

In the full proof of Lemma 18 in Appendix C of [16], we derive the relation between the security parameters of chain growth, common-prefix and chain quality shown above such that there is always sufficient time for an honestly encrypted input to reach the common-prefix; here we must carefully accomodate "time advantages" granted to the adversary. Firstly, observe that whenever there are empty slots between the chain tip and current slot, the adversary is granted time to decrypt the current session key without any progress towards finalization. Secondly, in the case of a leading adversarial block-span, the adversary is granted another head start in completing key extraction, as it can generate the block span prior to their associated slots and begin key extraction ahead of time.

▶ **Theorem 19 (Input fairness in (d, δ, Δ)-FairPoS).** *Let $\mathcal{A}$ be an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1 - \alpha)$ stake, where α be such that $\alpha(1 - f)\Delta = (1 + \epsilon)/2$ for active slot coefficient $f \in (0, 1]$ and some $\epsilon > 0$. Then, the probability that $\mathcal{A}$ makes the FairPoS violate the input fairness property falls exponentially with d.*

Theorem 19 is proven in Appendix C in [16] which relates the delay extraction parameter d with the security parameters of k-common-prefix, (τ, s)-chain growth and (μ, k)-chain quality.

6 Conclusion

We presented FairPoS, the first longest-chain, proof-of-stake protocol achieving input fairness against an adaptive adversary. When adopting the leader election procedure from Ouroborous Praos [19] or one that induces leader sequences (*i.e.* characteristic strings) consistent with the *dominant distribution* (Definition 1), FairPoS achieves input fairness in addition to the common-prefix, chain-growth and chain-quality properties of PoS [19]. We note that Kiayas *et al.* [30] provide tighter bounds for k-common prefix in PoS. Applying this updated analysis framework to security of FairPoS is planned as future work.

References

1 VDF Alliance. VDF Alliance Official Wiki. https://supranational.atlassian.net/wiki/spaces/VA/overview, 2022.

2 Avalanche. Apricot Phase Four: Snowman++ and Reduced C-Chain Transaction Fees. https://medium.com/avalancheavax/apricot-phase-four-snowman-and-reduced-c-chain-transaction-fees-1e1f67b42ecf, 2021.

3 Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930, 2018. doi:10.1145/3243734.3243848.

4 Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602, 2019. doi:10.1145/3319535.3363213.

5 Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. Maximizing extractable value from automated market makers. *arXiv preprint arXiv:2106.01870*, 2021. URL: https://arxiv.org/pdf/2106.01870.

6 Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Tardis: a foundation of time-lock puzzles in uc. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part III*, pages 429–459. Springer, 2021. doi:10.1007/978-3-030-77883-5_15.

7 Joseph Bebel and Dev Ojha. Ferveo: Threshold Decryption for Mempool Privacy in BFT networks. *Cryptology ePrint Archive*, 2022. URL: https://eprint.iacr.org/2022/898.

8 Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 431–448. Springer, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_28.

9 Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 260–290. Springer, Heidelberg, November 2020. doi:10.1007/978-3-030-64375-1_10.

10 Erica Blum, Aggelos Kiayias, Cristopher Moore, Saad Quader, and Alexander Russell. Linear consistency for proof-of-stake blockchains. *arXiv preprint arXiv:1911.10187*, 2019. URL: https://arxiv.org/abs/1911.10187.

11 Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings*, pages 236–254. Springer, 2000. URL: https://link.springer.com/content/pdf/10.1007/3-540-44598-6.pdf#page=248.

12 Jeffrey Burdges and Luca De Feo. Delay encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 302–326. Springer, 2021. doi:10.1007/978-3-030-77870-5_11.

13 Christian Cachin, Jovana Mićić, and Nathalie Steinhauer. Quick Order Fairness. *arXiv preprint arXiv:2112.06615*, 2021. URL: https://arxiv.org/abs/2112.06615.

14 Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future - A paradigm for sending secret messages to future (anonymous) committees. In *ASIACRYPT 2022, Part III*, LNCS, pages 151–180. Springer, Heidelberg, December 2022. doi:10.1007/978-3-031-22969-5_6.

15 Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. In *ASIACRYPT 2022, Part I*, LNCS, pages 651–680. Springer, Heidelberg, December 2022. doi:10.1007/978-3-031-22963-3_22.

16 James Hsin-yu Chiang, Bernardo David, Ittay Eyal, and Tiantian Gong. FairPoS: Input Fairness in Permissionless Consensus. `https://eprint.iacr.org/2022/1442`, 2023. Full paper version.

17 Dan Cline, Thaddeus Dryja, and Neha Narula. ClockWork: An Exchange Protocol for Proofs of Non Front-Running. In *The Stanford Blockchain Conference 2020*, 2020. URL: `https://www.media.mit.edu/publications/clockwork-an-exchange-protocol-for-proofs-of-non-front-running/`.

18 P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE Symposium on Security and Privacy*, pages 910–927. IEEE, 2020. `doi:10.1109/SP40000.2020.00040`.

19 Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018. `doi:10.1007/978-3-319-78375-8_3`.

20 Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 248–277. Springer, 2019. `doi:10.1007/978-3-030-34578-5_10`.

21 Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. Mcfly: Verifiable encryption to the future made practical. *Cryptology ePrint Archive*, 2022. URL: `https://eprint.iacr.org/2022/433`.

22 Andreas Erwig, Sebastian Faust, and Siavash Riahi. Large-scale non-interactive threshold cryptosystems through anonymity. Cryptology ePrint Archive, Report 2021/1290, 2021. URL: `https://eprint.iacr.org/2021/1290`.

23 Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 281–310. Springer, 2015. `doi:10.1007/978-3-662-46803-6_10`.

24 Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In *PKC 2022, Part I*, LNCS, pages 252–282. Springer, Heidelberg, May 2022. `doi:10.1007/978-3-030-97121-2_10`.

25 Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 332–354. Springer, Heidelberg, August 2001. `doi:10.1007/3-540-44647-8_20`.

26 Fredrik Kamphuis, Bernardo Magri, Ricky Lamberty, and Sebastian Faust. Revisiting transaction ledger robustness in the miner extractable value era. In *International Conference on Applied Cryptography and Network Security*, pages 675–698. Springer, 2023. `doi:10.1007/978-3-031-33491-7_25`.

27 Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. In *Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, pages 3–14, 2022. `doi:10.1145/3494105.3526239`.

28 Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, Strong Order-Fairness in Byzantine Consensus, 2021. URL: `https://eprint.iacr.org/2021/1465`.

29 Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Annual International Cryptology Conference*, pages 451–480. Springer, 2020. `doi:10.1007/978-3-030-56877-1_16`.

30 Aggelos Kiayias, Saad Quader, and Alexander Russell. Consistency of proof-of-stake blockchains with concurrent honest slot leaders. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 776–786. IEEE, 2020. `doi:10.1109/ICDCS47774.2020.00065`.

31 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017. doi:10.1007/978-3-319-63688-7_12.

32 Dahlia Malkhi and Pawel Szalachowski. Maximal Extractable Value (MEV) Protection on a DAG. *arXiv e-prints*, pages arXiv–2208, 2022. URL: https://arxiv.org/abs/2208.00940.

33 Peyman Momeni. Fairblock: Preventing blockchain front-running with minimal overheads. Master's thesis, University of Waterloo, 2022. URL: https://eprint.iacr.org/2022/1066.

34 Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology. Laboratory for Computer Science, 1996. URL: http://bitsavers.trailing-edge.com/pdf/mit/lcs/tr/MIT-LCS-TR-684.pdf.

35 Sri Aravinda Krishnan Thyagarajan, Tiantian Gong, Adithya Bhat, Aniket Kate, and Dominique Schröder. Opensquare: Decentralized repeated modular squaring service. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3447–3464, 2021. doi:10.1145/3460120.3484809.

36 Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. Sok: Decentralized finance (defi). *arXiv preprint arXiv:2101.08778*, 2021. URL: https://arxiv.org/abs/2101.08778.

Correlated-Output Differential Privacy and Applications to Dark Pools

James Hsin-yu Chiang
Aarhus University, Denmark

Bernardo David
IT University of Copenhagen, Denmark

Mariana Gama
COSIC, KU Leuven, Belgium

Christian Janos Lebeda
IT University of Copenhagen, Denmark
Basic Algorithms Research Copenhagen, Denmark

Abstract

In the classical setting of differential privacy, a privacy-preserving query is performed on a private database, after which the query result is released to the analyst; a differentially private query ensures that the presence of a single database entry is protected from the analyst's view. In this work, we contribute the first definitional framework for differential privacy in the *trusted curator setting* (Fig. 1); clients submit private inputs to the trusted curator, which then computes individual outputs privately returned to each client. The adversary is more powerful than the standard setting; it can corrupt up to $n-1$ clients and subsequently decide inputs and learn outputs of corrupted parties. In this setting, the adversary also obtains leakage from the honest output that is *correlated with a corrupted output*. Standard differentially private mechanisms protect client inputs but do not mitigate output correlation leaking arbitrary client information, which can forfeit client privacy completely. We initiate the investigation of a novel notion of *correlated-output differential privacy* to bound the leakage from output correlation in the trusted curator setting. We define the satisfaction of both standard and correlated-output differential privacy as *round differential privacy* and highlight the relevance of this novel privacy notion to all application domains in the trusted curator model.

We explore *round differential privacy* in traditional "dark pool" market venues, which promise privacy-preserving trade execution to mitigate front-running; privately submitted trade orders and trade execution are kept private by the trusted venue operator. We observe that dark pools satisfy neither classic *nor* correlated-output differential privacy; in markets with low trade activity, the adversary may trivially observe recurring, honest trading patterns, and anticipate and front-run future trades. In response, we present the first *round differentially private* market mechanisms that formally mitigate information leakage from all trading activity of a user. This is achieved with *fuzzy order matching*, inspired by the standard randomized response mechanism; however, this also introduces a liquidity mismatch as buy and sell orders are not guaranteed to execute pairwise, thereby weakening output correlation; this mismatch is compensated for by a round differentially private *liquidity provider* mechanism, which freezes a noisy amount of assets from the liquidity provider for the duration of a privacy epoch, but leaves trader balances unaffected. We propose oblivious algorithms for realizing our proposed market mechanisms with secure multi-party computation (MPC) and implement these in the Scale-Mamba Framework using Shamir Secret Sharing based MPC. We demonstrate practical, round differentially private trading with comparable throughput as prior work implementing (traditional) dark pool algorithms in MPC; our experiments demonstrate practicality for both traditional finance and decentralized finance settings.

2012 ACM Subject Classification Security and privacy → Privacy-preserving protocols

Keywords and phrases Differential Privacy, Secure Multi-party Computation, Dark Pools, Decentralized Finance

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.11

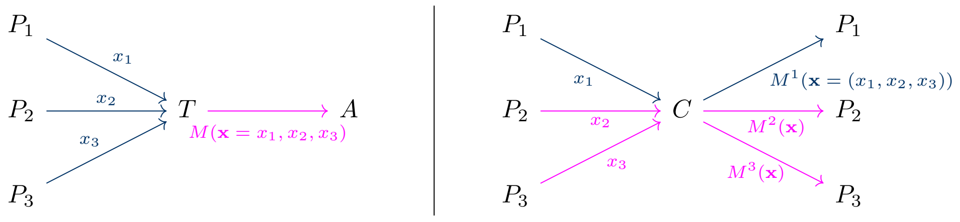


Figure 1 The standard model of differential privacy (L) vs. the trusted curator model (R). In this work, we contribute the first definitional framework for differential privacy in the trusted curator model (§3.1).

Related Version *Full Version*: `https://eprint.iacr.org/2023/943`

Supplementary Material *Software*: `https://github.com/maargama/CorrOutDP-darkpool`
archived at `swh:1:dir:3db7239f8dc2f11cc277b8c18dd37d5d5aae7f79`

Funding *James Hsin-yu Chiang*: Part of the work was supported by a DTU Compute scholarship.
Bernardo David: This work was supported by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B.
Mariana Gama: This work was supported by CyberSecurity Research Flanders with reference number VR20192203 and by the FWO under an Odysseus project GOH9718N.
Christian Janos Lebeda: This work was supported by the VILLUM Foundation grant 16582.

1 Introduction

In the standard differential privacy setting (Fig. 1, left), a single analyst (A) receives a query on private inputs from clients (P_1, P_2, P_3) computed by the trusted third party (T). A differentially private query protects the privacy of an input x_i submitted by client P_i. In the trusted curator model (Fig. 1, right), the curator C evaluates a function on all privately submitted inputs, $(y_1, y_2, y_3) \leftarrow M(x_1, x_2, x_3)$, and returns each output y_i privately to client P_i, which may be corrupted by the adversary. A (classically) differentially private mechanism M will protect the honest input x_1. However, if honest output $y_1 = M^1(x_1, x_2, x_3)$ and adversarial output $y_{i \neq 1} = M^i(x_1, x_2, x_3)$ are correlated, honest y_1 may be trivially inferred from an adversarial $y_{i \neq 1}$, breaking client privacy. In this work, we introduce *correlated-output differential privacy* (§3.3) to protect against such leakage to achieve client privacy in the setting of the trusted curator. The conjunction with standard differential privacy protecting inputs is defined as *round differential privacy* (§3.4), protecting the entire client transcript in each interaction round. In this model, the adversary can inject inputs to each round; *round differential privacy* insures that such a chosen-input attack has a bounded effect on the honest user's output. We highlight the investigation of round differentially private algorithms for general and specific application domains as a research question of independent interest. In this work, we investigate round differentially private market applications to prevent front-running in traditional or decentralized finance.

The term front-running originates from the notion of "getting in front" of pending trades. A party anticipating a large buy order may purchase the same asset first, as the pending large buy order will likely drive up the price of the asset; the front-running party can then sell the asset at a higher price following the execution of the large buy order. Front-running occurs whenever submitted trade orders that have yet to be executed are observable by the front-running adversary. In traditional finance, the presence of pending orders may be public or inferred from market order books. In decentralized finance, pending transactions are publicly gossiped across a peer-to-peer network. In both settings, front-running is prevalent.

In traditional finance, Dark Pool venues [25] promise the private execution of trades. Here, clients submit private orders to the venue operator, who then computes the execution of trades without leaking pending orders submitted by clients; pre-trade privacy ensures that pending orders remain private, whilst post-trade privacy protects the privacy of the trade execution. An initial attempt to provide both pre-trade and post-trade privacy would be to implement a market venue with a trusted curator role; in each round, parties can submit the trade orders privately to the curator, which then computes an optimal matching of submitted trade orders. The trade results are then privately output to each participating client. Since all communication between individual client and curator is private, no trade orders or trade outcomes are *explicitly* leaked. Removing the trusted curator role in dark pools with secure multi-party computation (MPC) has been recently explored in [6, 7, 12, 13], demonstrating practical run-times amenable to real-world trading volume.

However, we observe that *round differential privacy* is violated in classical order-matching algorithms run by dark pool venues; a trade execution always implies a counter-party, thus revealing the presence of another trade in the opposing direction (Lemmas 4 and 6); such leakage occurs because trade execution are necessarily *correlated* between different parties. In venues with low trade volume, such inferences may lead to practical attacks. Consider the Time-Weighted Average Price (TWAP)[1] trade, where a larger trade volume is scheduled as smaller trades over time to minimize price impact. If the periodic execution of such smaller trades is detected early, the remaining trade schedule can be anticipated and front-run. This motivates our investigation of market mechanisms with formal, privacy guarantees protecting the full transcript of interactions between trader and dark pool venue.

In this work, we contribute round-differentially-private (§4) market mechanisms for the trusted curator model. Here, the actual "trade", "no-trade" outcome for each order is determined by *sampling* a Bernoulli distribution biased towards the deterministically computed, optimal matching. However, since trades are filled or not filled based on independently sampling trade outcomes, there is no guarantee that each executed trade is matched with an equivalent volume in the opposing direction; therefore, a liquidity deficit may occur. Here, market makers or *liquidity providers* make up for liquidity deficits. To prevent liquidity providers from learning about the traded volume of a single user from their updated liquidity balances, a random, yet bounded amount of market maker liquidity is *frozen* to obtain round differential privacy; frozen liquidity is returned to the market makers at the end of each privacy epoch, that is sufficiently long for honest users to complete long-running trade strategies.

We instantiate our fuzzy order matching market mechanisms with oblivious MPC algorithms (§5) and implement these algorithms using the Scale-Mamba framework [3] with Shamir Secret Sharing based MPC; history has shown that dark pool venue operators frequently exploit confidential order flow information [20, 21, 19], thus motivating us to demonstrate practical feasibility of distributing round differential private market mechanisms across MPC committees in lieu of a trusted operator. We show that our algorithms not only satisfy stronger privacy guarantees than considered in previous work, they also achieve high order throughput appropriate for most real-world settings. In fact, even with the additional overhead induced by round differentially private algorithms, we obtain runtimes

[1] `https://en.wikipedia.org/wiki/Time-weighted_average_price`

comparable to the algorithms presented in prior work implementing traditional dark pools in MPC [12]. Finally, we emphasize that our fair market mechanism designs are applicable to both traditional dark pool venue operators and decentralized finance. Our fair markets can be instantiated in privacy-preserving smart contract frameworks realized by an MPC committee and privacy-preserving ledger, most recently demonstrated by Baum et al. in [4] with minimal complexity overhead; here, trade execution is settled in private on a public ledger.

1.1 Related Work

Differentially private markets. Chitra *et al.* [11] propose a *Uniform Random Execution* algorithm which permutes and splits submitted trades in a randomized, differentially private manner. We note that [11] does not offer output or *post-trade* privacy; all executed trades are seen by the adversary. Thus, this approach does not contribute to our goal of protecting the privacy of long-running trader strategies performed over multiple rounds.

Dark pool markets. Recent proposals [6, 7, 12, 13] have convincingly demonstrated that the role distribution of the dark pool operator can be instantiated in practice with multi-party computation (MPC) to prevent abuse of private order information. Still, these works also do not consider the entirety of information flow leaking from all honest trader activity; Firstly, adversarial outputs reveal information about privately submitted honest inputs (Lemma 4) and secondly, outputs are correlated, such that an adversary also obtains information about honest outputs (Lemma 6). In the decentralized finance setting, homomorphic encryption has been proposed to aggregate orders obliviously [24]; however, since all inputs are encrypted to the same public key, any subsequent decryption to reveal the aggregated order will leak the privacy of any single trade, if all but one client has been corrupted.

Differential privacy and MPC. Whilst differentially private mechanisms have been implemented in MPC, these works do not consider privacy over the full, individual transcript in the trusted curator model (§3.1), where clients submit private inputs and receive private outputs. Instead, the MPC output is a single query result computed over inputs from a private database. Here, the returned query is not considered private. The main use-case is generating differentially private machine learning models over private data with MPC [22, 1, 27, 23].

2 Preliminaries

Differential privacy. Differential privacy was introduced in [14] as a technique for quantifying the privacy guarantees of a mechanism. A central concept is the definition of neighbouring datasets which are denoted $x \sim x'$. Intuitively, this definition is used to capture the information we want to protect. Typically x and x' are identical except for the data about one individual. We formally define neighbouring inputs in our setting of the trusted curator in Section 3.1. Differential privacy is a restriction on how much the output distribution of a mechanism can change between any neighbouring input datasets.

▶ **Definition 1** (**(ε, δ)-DP**). *A randomized mechanism $\mathcal{M}$ satisfies (ε, δ)-differential privacy if for all pairs of neighbouring datasets $x \sim x'$ and all sets of outputs $\mathcal{S}$ we have:*

$$\Pr[\mathcal{M}(x) \in \mathcal{S}] \leq \exp(\varepsilon) \cdot \Pr[\mathcal{M}(x') \in \mathcal{S}] + \delta$$

Multi-party computation. Multi-party computation is a cryptographic technique that allows a set of n mistrustful parties to calculate a function of their own private inputs without revealing them. We consider an MPC protocol based on Shamir secret sharing, where a secret value s is shared by giving each party i the evaluation $f(i)$ of a polynomial f of degree t and coefficients in $\mathbb{F}_p$ such that $f(0) = s$. The protocol assumes an honest majority, i.e., $t < n/2$, and it is actively secure with an abort, meaning that a malicious party deviating from the protocol is caught with overwhelming probability and the honest parties abort the protocol when this happens. In this work, we use Scale-Mamba [3], a framework that implements various MPC protocols in the preprocessing model. In this methodology, the computation has a preprocessing phase where input independent data is generated. This data is then used in the input dependent online phase, where the desired computation over private inputs is performed.

3 Round differential privacy in the trusted curator model

In this section, we first formalize the round-based trusted curator model (§3.1). Round differential privacy is defined over two distinct notions: (1) Input differential privacy (§3.2) protects the honest input, and mirrors the classic notion of differential privacy over a private database. (2) Correlated output differential privacy (§3.3) protects leakage from correlated outputs, and represents a novel privacy notion of general interest for all application domains. Round differential privacy (§3.4) is defined over (1) and (2); here, we also formalize multi-round differential privacy, to protect the entire honest transcript over multiple interaction rounds in the trusted curator model.

3.1 The trusted curator

We first define our proposed notions of privacy in the "trusted curator" model (Fig. 1), which can then seamlessly be applied to the setting of secure multi-party computation. The trusted curator C interacts with parties $P_1, ..., P_n$, which are assumed to have established private, authenticated communication links with the trusted curator. Interaction proceeds in *rounds*, each consisting of the following phases.

1. **Input phase** All parties send their individual inputs to the trusted curator C, which obtains the input set $x_1, ..., x_n$ from parties $P_1, ..., P_n$ respectively.
2. **Evaluation phase** Upon receiving all inputs, the trusted curator locally computes a known algorithm M over inputs received in the input phase: namely $\mathbf{y} \leftarrow \mathbf{M}(\mathbf{x})$, where $\mathbf{x} = (x_1, ..., x_n)$ and $\mathbf{y} = (y_1, ..., y_n)$. Further, curator C is assumed to have access to randomness to evaluate randomized algorithms.
3. **Output phase** The trusted curator privately sends each output element y_i in $\mathbf{y}$ to party P_i, and enters the input phase again. Any "public output" y_{pub} is encoded in each individual output; $\forall i \in [n] : y_{\mathsf{pub}} \in y_i$.

Client corruption. The adversary $\mathcal{A}$ can statically corrupt up to $n - 1$ clients, upon which it decides what inputs the corrupted clients submit in each interaction round. The adversary decides corrupted inputs and observes the output for each corrupted client returned from the trusted curator; the adversary cannot corrupt the curator itself. We denote the adversarial output view from a round evaluating mechanism $\mathbf{M}$ on round inputs $\mathbf{x}$ as $\mathbf{M}^{\mathcal{A}}(\mathbf{x})$.

Public outputs. We permit the trusted curator to also return public outputs; naturally, any public output is part of the adversarial output view $\mathbf{M}^{\mathcal{A}}(\mathbf{x})$.

Privacy against the network adversary. We assume that the *physical presence* of a party in each round is observable by the network adversary. Since obfuscating the active participation across the network may be challenging, we assume parties to be physically online and to participate in each round, but permit them to submit **dummy inputs**, allowing for passive participation and obfuscating the *logical presence* of a party in a given round. Without dummy inputs, the physical presence of a party will always leak the presence of a logical input contributed by a party to the computation by the trusted curator; in the setting of privacy-preserving markets, for example, the network adversary would learn that a party is submitting some trade in a given round.

Further, we assume that parties can anonymously submit inputs to the trusted curator via techniques such as mixnets [9, 8], thereby hiding their identity from the network adversary. In practice, parties can delegate the physical interaction with the trusted curator model in each round to trusted servers, and only need to come online when they wish to forward a valid, non-dummy input.

Group privacy. We highlight that *individual* differential privacy guarantees introduced in the subsequent section naturally imply *group privacy*; a mechanism protecting the presence of a single client can do so for multiple clients, consuming equal privacy budget amounts for each additional group member.

3.2 Differential privacy for inputs

In the standard setting of differential privacy, an analyst performs a query on a private database and the result of the query is released to the analyst; a differentially private query bounds how much the analyst output distribution changes, upon adding or removing an entry in the private database.

We adapt the classic notion of differential privacy to the setting of the trusted curator. Instead of protecting private database entries, we first wish to protect inputs submitted by honest clients. We introduce a dummy value that allows clients to have no impact on a given round. Thus, the following definition of neighbouring input vectors follows directly from the standard definition of neighbouring databases under the add/remove relation in the classic setting.

▶ **Definition 2 (neighboring input vectors).** *Input vectors $\mathbf{x} = (x_1, ..., x_n)$ and $\mathbf{x}' = (x'_1, ..., x'_n)$ of equal length are neighboring, denoted as $x \sim x'$, if the following holds true;*

$$\exists i \in [n] : x_j = x'_j \text{ for all } j \neq i \text{ and } (x_i = \textit{dummy} \text{ or } x'_i = \textit{dummy})$$

For a randomized algorithm $\mathbf{M}$ evaluated on input vector $\mathbf{x}$, let $\mathbf{M}^{\mathcal{A}}(\mathbf{x}) = \{Y_j\}_{j\in\mathcal{A}}$ denote output distributions observed by corrupted clients. Then, the following definition follows directly from the standard notion [15] of differential privacy where we consider the input vector as the private database on which the query $\mathbf{M}$ is performed and the adversary obtains the output view $\mathbf{M}^{\mathcal{A}}(\mathbf{x})$ of all corrupted parties. Note that there is no restriction on the output distribution seen only by the honest user.

▶ **Definition 3 ((ε, δ)-input DP).** *For an evaluation of (ε, δ)-input differentially private algorithm $\mathbf{M}$ in the trusted curator model over neighboring private input vectors $\mathbf{x} \sim \mathbf{x}'$, the following must hold for any adversarially observable output event $\mathcal{S}^{\mathcal{A}}$.*

$$\Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}) \in \mathcal{S}^{\mathcal{A}} \,] \leq \exp(\varepsilon) \cdot \Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}') \in \mathcal{S}^{\mathcal{A}} \,] + \delta$$

As we will see in Section 3.3, input differential privacy is necessary, but insufficient to protect both in- and output of an honest client in the trusted curator round. Whilst Definition 3 protects the privacy of a user input, it does not guarantee that the honest output remains private. This motivates *correlated-output* differential privacy, introduced in the subsequent section Section 3.3. Again, the standard setting of differential privacy does not consider the privacy of the query output, as there is only a single query result released publicly or to the adversarial analyst.

▶ **Lemma 4.** *Dark Pools violate (ε, δ)-input differential-privacy for any $\delta < 1$.*

Proof (Sketch). A dark pool venue operator can be idealized as a trusted curator which privately receives trade orders from clients. Upon evaluating the market algorithm in private, it privately outputs trade executions to clients. Assume an honest user submits the only buy order and the corrupted client submitting a sell order observes that its trade order is executed. Any change in the honest counter-party's privately submitted buy order cancels the matching of this order pair, observable to the adversary with probability 1, thereby violating Definition 3. ◀

Adversarially chosen inputs. Note that input differential privacy in Definition 3 naturally protects against *chosen input attacks*; informally, such an attack permits the adversary to change its inputs and observe induced effects on its output distributions to learn something about honest inputs. However, note that (ε, δ)-input DP applies equal privacy guarantees to *any input* submitted to the trusted curator. Thus, for appropriately chosen privacy parameters, a chosen input attack on an (ε, δ)-input DP mechanism will not reveal meaningful information to the adversary, as its chosen input perturbation will not induce a sufficiently observable effect on its output distributions.

3.3 Differential privacy for correlated outputs

In contrast to prior work, where a single output is returned from a differentially-private mechanism, we must protect the privacy of outputs that are returned from the trusted curator to individual clients over private channels. Even if input differential privacy protects honest inputs, the individual outputs returned to clients may still be strongly correlated, potentially allowing honest outputs to be inferred from corrupted ones.

▶ **Definition 5 ((ε, δ)-correlated-output DP).** *For an evaluation of (ε, δ)-correlated output differentially private algorithm $\mathbf{M}$ in the trusted curator model over fixed input vector $\mathbf{x}$, the following must hold for any adversarial output event $\mathcal{S}^{\mathcal{A}}$ and any honest output event $\mathcal{S}^h$.*

$$\Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}) \in \mathcal{S}^{\mathcal{A}} \mid \mathbf{M}^h(\mathbf{x}) \in \mathcal{S}^h \,] \leq \exp(\varepsilon) \cdot \Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}) \in \mathcal{S}^{\mathcal{A}} \mid \mathbf{M}^h(\mathbf{x}) \notin \mathcal{S}^h \,] + \delta$$

Definition 5 is interpreted as follows; for any set of inputs and two different honest output events ($\mathbf{M}^h(\mathbf{x}) \in \mathcal{S}^h$ vs. $\mathbf{M}^h(\mathbf{x}) \notin \mathcal{S}^h$), the output distribution $\mathbf{M}^{\mathcal{A}}(\mathbf{x})$ of the adversary remains (ε, δ)-similar. In other words, any change in the honest output can only have a bounded effect on the adversarially observable output distribution.

We highlight an immediate consequence of Definition 5 for economic applications; a correlated-output DP mechanism cannot distribute funds to all clients where the supply of output funds is known or public; an adversary corrupting $n-1$ clients can trivially infer the funds privately output to the single honest client by just aggregating its own outputs and observing the difference to the total supply. Thus;

▶ **Lemma 6.** *Economic mechanisms evaluated in the trusted curator model which allocate a fixed supply of "assets" over client outputs violate (ε, δ)-correlated-output differential privacy for any $\delta < 1$.*

Overcoming this is not straight-forward, as financial applications cannot be allowed to arbitrarily mint or create funds out of thin air. We overcome these constraints by performing *fuzzy matching* of orders and temporarily freezing funds to achieve correlated-output differential privacy in rDP-volume-match (Section 4.1) and rDP-double-auction (Section 4.2).

Applications with correlated outputs. We argue there exist many applications in the trusted curator setting which require correlated outputs; most closely related to this work are economic applications which govern the private allocation of finite resources, which include auctions, markets, financial derivatives and other economic contracts.

3.4 Single-round & Multi-round privacy

Since (ε, δ)-input DP and (ε, δ)-correlated-output DP protect different parts of the honest round transcript, we must formally consider two separate privacy budgets which are consumed with each interaction round in the trusted curator model. We define differential privacy for *each interaction round* with the trusted curator as follows.

▶ **Definition 7 (Round-DP).** *The evaluation of a mechanism that satisfies $(\varepsilon^{in}, \delta^{in})$-input DP and $(\varepsilon^{out}, \delta^{out})$-correlated-output DP is $(\varepsilon^{in}, \delta^{in})$-$(\varepsilon^{out}, \delta^{out})$-round differentially private.*

Definition 7 implies that the privacy of input and outputs may be parameterized *independently*. Indeed, this permits the trade-off between utility and privacy for different parts of the honest transcript to be decided separately; the input to an evaluation round may require a higher degree of privacy than the returned output or vice versa.

Multi-round privacy. When applying differential privacy to queries on a static database, m instances of $(\varepsilon_i, \delta_i)$-differentially private queries taken together are $(\varepsilon_1 + ... + \varepsilon_m, \delta_1 + ... + \delta_m)$ differentially private using basic composition [15] with tighter bounds known using advanced composition [17]. However, in the trusted curator model, the curator accepts fresh inputs in each interaction round, allowing us to consider each round input as disjoint, private data. We define multi-round differential privacy as the sensitivity of the adversarial output view over m rounds to changes in inputs submitted and outputs received by a single client.

▶ **Definition 8 (m-round-DP).** *Let the adversarial output view over m interaction rounds between n-clients and the trusted curator be given as $\mathbf{M}_1^{\mathcal{A}}(\mathbf{x}_1), \dots, \mathbf{M}_m^{\mathcal{A}}(\mathbf{x}_m)$. Then, we define this m-round transcript as;*

$$\mathbf{M}_{mul}^{\mathcal{A}}(\bar{\mathbf{x}}) = (\mathbf{M}_1^{\mathcal{A}}(\mathbf{x}_1), \dots, \mathbf{M}_m^{\mathcal{A}}(\mathbf{x}_m)) \text{ where } \bar{\mathbf{x}} = (\mathbf{x}_1, \dots, \mathbf{x}_m) \text{ are round-specific client inputs.}$$

Let m-round client inputs $\bar{\mathbf{x}} = (\mathbf{x}_1, ..., \mathbf{x}_m)$ and $\bar{\mathbf{x}}' = (\mathbf{x}'_1, ..., \mathbf{x}'_m)$ be neighboring if they only differ in inputs submitted by a single client throughout the m rounds;

$$\exists! \textit{client } i : \forall \textit{round } r \in [m] : \big(\mathbf{x}_r = \mathbf{x}'_r \big) \textit{ or } \big(\mathbf{x}_r \sim \mathbf{x}'_r \textit{ where } \mathbf{x}_r(i) \neq \mathbf{x}'_r(i) \big)$$

Further, we denote an m-round output event for the adversary and honest client as $\mathcal{S}_{mul}^{\mathcal{A}} = \mathcal{S}_1^{\mathcal{A}}, ..., \mathcal{S}_m^{\mathcal{A}}$ and $\mathcal{S}_{mul}^{h} = \mathcal{S}_1^{h}, ..., \mathcal{S}_m^{h}$ respectively.

The m-round interaction is $(\varepsilon^{in}, \delta^{in})$-$(\varepsilon^{out}, \delta^{out})$-$m$-round differentially private if for any two neighbouring m-round inputs $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$, any adversarial and honest m-round events $\mathcal{S}^{\mathcal{A}}_{mul}$ and $\mathcal{S}^{h}_{mul}$, the following holds true;

$$\begin{aligned}&\Pr[\,\mathbf{M}^{\mathcal{A}}_{mul}(\bar{\mathbf{x}}) \in \mathcal{S}^{\mathcal{A}}_{mul}\,]\\ \leq\ &\exp(\varepsilon^{in}) \cdot \Pr[\,\mathbf{M}^{\mathcal{A}}_{mul}(\bar{\mathbf{x}}') \in \mathcal{S}^{\mathcal{A}}_{mul}\,] + \delta^{in} \qquad (a)\end{aligned}$$

$$\begin{aligned}&\Pr[\,\mathbf{M}^{\mathcal{A}}_{mul}(\bar{\mathbf{x}}) \in \mathcal{S}^{\mathcal{A}}_{mul} \mid \mathbf{M}^{h}_{mul}(\bar{\mathbf{x}}) = \mathcal{S}^{h}_{mul}\,]\\ \leq\ &\exp(\varepsilon^{out}) \cdot \Pr[\,\mathbf{M}^{\mathcal{A}}_{mul}(\bar{\mathbf{x}}) \in \mathcal{S}^{\mathcal{A}}_{mul} \mid \mathbf{M}^{h}_{mul}(\bar{\mathbf{x}}) \neq \mathcal{S}^{h}_{mul}\,] + \delta^{out} \qquad (b)\end{aligned}$$

The following theorem relates single-round DP (def. 7) with m-round DP (def. 8), allowing us to achieve multi-round privacy from sequential interaction rounds between clients and the trusted curator.

▶ **Theorem 9 (m-round composition (updated 2)).** *Let there be m consecutive interaction rounds with n clients and the trusted curator. In each round, the trusted curator evaluates round-specific algorithms $\mathbf{M}_1, \ldots, \mathbf{M}_m$ that are run independently and are $(\varepsilon^{in}_1, \delta^{in}_1)$-$(\varepsilon^{out}_1, \delta^{out}_1)$, ..., $(\varepsilon^{in}_m, \delta^{in}_m)$-$(\varepsilon^{out}_m, \delta^{out}_m)$ round differentially private and evaluated on round-specific input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$. Then, the m-round evaluation is*

$$\Big(\sum_{j\in[m]} \varepsilon^{in}_j,\ \sum_{j\in[m]} \delta^{in}_j\Big) - \Big(\sum_{j\in[m]} \varepsilon^{out}_j,\ \sum_{j\in[m]} \delta^{out}_j\Big)$$

m-round differentially private.

Proof. We use the basic version of the adaptive composition theorem for the proof. Since the inputs in each round are either equal or neighboring, the m-round notion of input DP (Eq. (a) in Def. 8) follows directly from applying the Composition Theorem for approximate DP (see [26, Theorem 22]).

Towards satisfying the m-round notion of correlated-output DP (Eq. (b) in Def. 8) notice that for each round we can define the event $\mathcal{S}^h$ that the honest output agrees with $\mathcal{S}^h_{\mathsf{mul}}$. Definition 5 then tells us that each round is $(\varepsilon_j, \delta_j)$-indistinguishable. Similar to the input DP we can use the Composition Theorem to get guarantees for the m-round correlated-output DP. ◀

4 Round differentially private market mechanisms

We propose round differentially private market mechanisms in the trusted curator model. These include volume matching of orders, where a batch of buy and sell orders (§4.1) are matched at a given exchange rate determined at an external reference market, and double auctions (§4.2), where buy and sell orders also feature price limits, such that a clearing price must first be computed for each round before orders can be matched. Following a gentle introduction, each algorithm is formally proven to satisfy round differential-privacy.

To realize any meaningful notion of privacy in practice, we later distribute the trusted curator by means of secure multi-party computation (MPC) in section §5.

4.1 Round-DP volume matching

In volume matching, the exchange rate is pre-determined by an external reference rate. A trader only chooses to submit a sell, buy or to abstain from the round with a dummy order. We introduce a $(\varepsilon^{\mathsf{in}}, \delta^{\mathsf{in}})$-$(\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}})$-round differentially private volume matching algorithm named rDP-Volume-match, overcoming the privacy limitations of the traditional dark pool setting, where a matched buy and sell order pair implies leaking the presence of an order execution to the counter-party and thereby violates both input- and correlated-output differential privacy (Lemmas 4 and 6). We overcome this privacy challenge with the following two phases in the rDP-Volume-match algorithm (Fig. 2).

1. Fuzzy order matching. In the first phase of rDP-volume-match, orders are matched in a "fuzzy" manner; following a preliminary, deterministic matching step which maximizes number of trades, each final trade output (trade/no-trade) for each client is sampled independently from a distribution parameterized by $\varepsilon^{\mathsf{in}}$ and biased towards the preliminary matching result; we adapt this technique from the standard randomized response mechanism [15, 28], that is both $(\varepsilon^{\mathsf{in}}, 0)$-input and $(0, 0)$-correlated output differentially private; the latter property arises naturally from the independent sampling of outputs which occurs in randomized response. Randomized, fuzzy matching of orders also implies that the final aggregate exchange of assets may not sum to zero; in any given round, the total buy volume may not equal the total sell volume. We handle this mismatch in the second phase of rDP-volume-match.

2. Liquidity compensation. We introduce a *liquidity provider*, which compensates for the mismatch between buy and sell volume; however, without any additional treatment, the adversary corrupting $n-1$ traders *and* the liquidity provider can trivially learn the honest output from the implied flow of assets between corrupt and honest parties (e.g. output correlation). To ensure that the corrupt liquidity provider's output is correlated-output differentially private, a randomized amount of its liquidity is frozen; here, the parameterization of rDP-Volume-match permits the choice of an upper limit (ρ^{max}) on frozen volumes of both the risky and numeraire asset types, thereby bounding the opportunity cost imposed on the liquidity provider. We define a *privacy epoch* over multiple rounds in Definition 10, during which the privacy guarantees of rDP-volume-match hold; if the frozen liquidity is later returned to the liquidity provider, round differential privacy is no longer guaranteed. In practice, we argue that it is acceptable to guarantee round differential privacy for a bounded number of rounds, during which honest users can complete their multi-round trading strategy without front-running interference. For privacy guarantees to hold indefinitely, assets would have to burned. Note that assets are never minted, preserving the integrity of their supply. We also note that, in principle, multiple liquidity providers could participate in each round of rDP-Volume-Match; we model a single liquidity provider to simplify exposition and formal proofs.

Next, we detail and motivate steps of rDP-volume-match and refer to Fig. 2 for a formal description of the algorithm.

Orders in rDP-Volume-match. Let a valid, privately submitted trade order be the tuple (b, s, id), where b and s represent buy and sell bits respectively, and id is the trader identifier. Thus, let $(b, s) \in [(1, 0), (0, 1), (0, 0)]$ represent a buy, sell and dummy order respectively. We fix buy and sell unit volumes such that a single sell and buy order always match in exchanged asset value.

1a. Deterministic matching (1a. in Fig. 2). Let the number of orders sent to the trusted curator by n clients be $\mathbf{x} = \{(b, s, \mathsf{id}_1), ..., (b, s, \mathsf{id}_n)\}$. Then, the maximum possible number of matches between buy $(b, s) = (1, 0)$ and sell $(b, s) = (0, 1)$ orders is computed, which is

```
rDP-Volume-match. Following inputs are assumed to be well-formed for simplicity.
  - Each trader $\mathcal{P}_i^{\mathsf{trd}} \in (\mathcal{P}_1^{\mathsf{trd}}, ..., \mathcal{P}_n^{\mathsf{trd}})$ inputs trade order $(b_i, s_i, \mathsf{id}_i)$.
  - The liquidity provider $\mathcal{P}^{\mathsf{liq}}$ inputs liquidity amounts $(x_0^{\mathsf{liq}}, x_1^{\mathsf{liq}})$.
  - The privacy parameters $\varepsilon^{\mathsf{in}}, \varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}} > 0$ and $\rho^{\mathsf{max}}$.

  1. Fuzzy order matching
    1a. Deterministic matching of buy & sell orders
      - Aggregate number of buy and sells; $\forall i$: $B \leftarrow B + b_i$, and $S \leftarrow S + s_i$
      - Determine the number of matched pairs; $u \leftarrow \mathsf{min}(B, S)$.
      - Match $u$ pairs of buy and sell orders;
        Let $\mathsf{match}_i \in \{0, 1\}$ indicate a match for party $P_i$ for $i \in [n]$.
    1b. Randomized response over order matches
      - $\forall i$: If $\mathsf{match}_i$, sample $\mathsf{trade}_i \leftarrow\!\!\$\ e^{\varepsilon^{\mathsf{in}}}/(1 + e^{\varepsilon^{\mathsf{in}}})$, otherwise $\mathsf{trade}_i \leftarrow\!\!\$\ 1/(1 + e^{\varepsilon^{\mathsf{in}}})$
        - If $\mathsf{trade}_i$ is true and $(b_i, s_i) \neq (0, 0)$;
          * Return $(b_i^{\mathsf{out}} = 1, s_i^{\mathsf{out}} = 0, \mathsf{id}_i)$ to $\mathcal{P}_i^{\mathsf{trd}}$ for a buy order $(b_i = 1, s_i = 0)$
          * Return $(b_i^{\mathsf{out}} = 0, s_i^{\mathsf{out}} = 1, \mathsf{id}_i)$ to $\mathcal{P}_i^{\mathsf{trd}}$ for a sell order $(b_i = 0, s_i = 1)$
        - Else return $(b_i^{\mathsf{out}} = 0, s_i^{\mathsf{out}} = 0, \mathsf{id}_i)$ to $\mathcal{P}_i^{\mathsf{trd}}$

  2. Liquidity compensation
    2a. Liquidity compensation for sampled trades
      - Aggregate final buy and sell volumes; $\forall i$: $o^b \leftarrow o^b + b_i^{\mathsf{out}}$, and $o^s \leftarrow o^s + s_i^{\mathsf{out}}$
      - Determine the liquidity mismatch; $\Delta_0 \leftarrow (o^s - o^b)$ and $\Delta_1 \leftarrow -\Delta_0$
    2b. Randomized liquidity freezing
      - Sample frozen volumes; $\forall t \in \{0, 1\} : \rho_t \leftarrow\!\!\$\ P_{\mathsf{frz}}$ (Param. by $\rho^{\mathsf{max}}, \varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}}$ in Eq. 4).
      - Return liquidity amounts $(y_0^{\mathsf{liq}}, y_1^{\mathsf{liq}}) \leftarrow (x_0^{\mathsf{liq}} + \Delta_0 - \rho_0\,,\ x_1^{\mathsf{liq}} + \Delta_1 - \rho_1)$ to $\mathcal{P}^{\mathsf{liq}}$.
      - Freeze $(\rho_0, \rho_1)$, return to $\mathcal{P}^{\mathsf{liq}}$ at the end of the privacy epoch.
```

Figure 2 RDP-Volume-match algorithm.

simply the smaller of the number of buy b and sell s orders. Let the result of the deterministic matching phase be the bit array $\mathsf{match} = (\mathsf{match}_1, ..., \mathsf{match}_n)$, where bit match_i indicates if the i'th submitted order was matched (1) or not (0). Once the total number of preliminary matched pairs is computed, they are assigned randomly to the non-dummy orders; dummy orders are never matched.

1b. Randomized response over matches (1b. in Fig. 2). Here, we apply the standard randomized response mechanism [15, 28] to determine whether a *trade* or *no-trade* is returned to the trader who submitted a valid trade order; for each bit in array match where $\mathsf{match}_i = 1$, the probability of the final trade_i bit equaling 1 or 0 is given by;

$$\begin{aligned} &\Pr[\,\mathsf{trade}_i = 1 \,|\, \mathsf{match}_i = 1\,] = e^{\varepsilon^{\mathsf{in}}}/(1 + e^{\varepsilon^{\mathsf{in}}}) \\ &\Pr[\,\mathsf{trade}_i = 0 \,|\, \mathsf{match}_i = 1\,] = 1/(1 + e^{\varepsilon^{\mathsf{in}}}) \end{aligned} \tag{1}$$

Conversely, for each bit $\mathsf{match}_i = 0$ in match and in the case that party i did not submit a dummy order, the probability of the final trade_i outcome being sampled as 1 or 0 is given by;

$$\begin{aligned} &\Pr[\,\mathsf{trade}_i = 1 \,|\, \mathsf{match}_i = 0\,] = 1/(1 + e^{\varepsilon^{\mathsf{in}}}) \\ &\Pr[\,\mathsf{trade}_i = 0 \,|\, \mathsf{match}_i = 0\,] = e^{\varepsilon^{\mathsf{in}}}/(1 + e^{\varepsilon^{\mathsf{in}}}) \end{aligned} \tag{2}$$

Thus, for parties submitting valid, non-dummy trades, each of the final trading results in array $\mathsf{trade} = [\mathsf{trade}_1, ..., \mathsf{trade}_n]$ is obtained from independently sampling from distributions Eq. 1 or 2 according to the match_i bit output from the deterministic matching subroutine [1a]. Trader outputs are given by the array $[(b_1^{\mathsf{out}}, s_1^{\mathsf{out}}, \mathsf{id}_1), ..., (b_n^{\mathsf{out}}, s_n^{\mathsf{out}}, \mathsf{id}_n)]$, where each entry $(b_i^{\mathsf{out}}, s_i^{\mathsf{out}}, \mathsf{id}_i)$ returned to party i indicates whether a buy $(b_i^{\mathsf{out}}, s_i^{\mathsf{out}}) = (1, 0)$, sell $(b_i^{\mathsf{out}}, s_i^{\mathsf{out}}) = (0, 1)$ or no trade $(b_i^{\mathsf{out}}, s_i^{\mathsf{out}}) = (0, 0)$ was executed;

We emphasize that a trade can only be executed if a non-dummy order was submitted at the beginning of the round, and in the same direction (sell or buy) as intended by the trader. Dummy orders always return $(b^{\mathsf{out}}, s^{\mathsf{out}}) = (0, 0)$ as output; the fuzzy matching is only applied to valid, non-dummy orders only, and thus the trading "interface" remains the same as in traditional volume matching algorithms; a trade order is either filled or not at all.

2a. Liquidity compensation for sampled trades (2a. Fig. 2). Fuzzy matching of orders via randomized response implies that traded volumes from step [1b] in rDP-volume-match do not match precisely; for the trade outputs $[(b_1^{\mathsf{out}}, s_1^{\mathsf{out}}, \mathsf{id}_1), ..., (b_n^{\mathsf{out}}, s_n^{\mathsf{out}}, \mathsf{id}_n)]$, the following can occur;

$$\sum_{i \in [n]} s_i^{\mathsf{out}} \neq \sum_{i \in [n]} b_i^{\mathsf{out}}$$

Since sells and buys may not cancel out, we introduce the presence of a *liquidity provider*, which compensates for this mismatch in traded asset liquidity. Then, the amount of the numeraire asset (Δ_0) and risky asset (Δ_1) provided $(\Delta < 0)$ or received $(\Delta > 0)$ by the liquidity provider is given as;

$$\Delta_0 = -\sum_{i \in [n]} s_i^{\mathsf{out}} - b_i^{\mathsf{out}} \qquad \Delta_1 = \sum_{i \in [n]} s_i^{\mathsf{out}} - b_i^{\mathsf{out}} \tag{3}$$

The liquidity provider compensates for this liquidity imbalance resulting from fuzzy matching; its initial balances $(x_0^{\mathsf{liq}}, x_1^{\mathsf{liq}})$ are updated to $(x_0^{\mathsf{liq}} + \Delta_0, x_1^{\mathsf{liq}} + \Delta_1)$; however, note that any change in the honest user's trade execution will affect Δ_0, Δ_1 with probability 1, observable by the corrupted liquidity provider and violating correlated-output differential privacy (Def. 5); *relaxing* the correlation between the final exchange of assets and the update in funds observed by the liquidity provider can only imply the *minting* or *removal* of funds in the round outputs.

We propose a compromise, which is a randomized mechanism to *freeze liquidity*, protecting the privacy of traders for the m-round duration that the liquidity remains frozen; we call this a privacy epoch (Def. 10). Our algorithm DP-volume-match refrains from minting, preserving the integrity of the underlying asset types.

2b. Randomized liquidity freezing (2b. in Fig. 2). (L8 in Figure 6). The liquidity provider inputs $(x_0^{\mathsf{liq}}, x_1^{\mathsf{liq}})$ amounts of numeraire (0) and risky (1) asset to a given round, and is returned updated reserve balances $(y_0^{\mathsf{liq}}, y_1^{\mathsf{liq}}) = (x_0^{\mathsf{liq}} + \Delta_0 - \rho_0, x_1^{\mathsf{liq}} + \Delta_1 - \rho_1)$, where (ρ_0, ρ_1) is the volume of assets (0) and (1) frozen in the given round and returned at the end of the privacy epoch, chosen to be sufficiently long to protect a common trading strategies executed over multiple rounds.

Note that it would be easy to freeze liquidity with perfect privacy if we had unbounded liquidity. The liquidity provider could provide n units of each asset in every round and liquidity would be frozen such that $(y_0^{\mathsf{liq}}, y_1^{\mathsf{liq}}) = (x_0^{\mathsf{liq}} - n, x_1^{\mathsf{liq}} - n)$. However, the required liquidity would not be feasible for large n. Our mechanism instead provides a trade-off

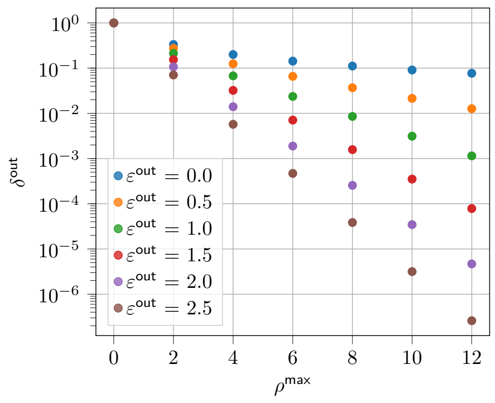


Figure 3 We plot selected parameterizations of P_{frz} in Eq. 4. The choice of parameters represents a trade-off between degree of privacy ($\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}}$) and frozen funds ($\rho^{\mathsf{max}}$).

between privacy and frozen liquidity. In each round we sample $\rho_0 \in [0, \rho^{\mathsf{max}}]$ and set $\rho_1 = \rho^{\mathsf{max}} - \rho_0$. We give the probability mass function P_{frz} from which ρ_0 is sampled in Equation (4); this distribution is parameterized by a maximum amount of frozen liquidity $\rho^{\mathsf{max}} \geq 1$ in the round, and correlated-output differential privacy parameters $\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}}$.

$$P_{\mathsf{frz}}(\rho_0) = \begin{cases} \delta^{\mathsf{out}} \cdot \exp(\varepsilon^{\mathsf{out}} \cdot \rho_0) & \rho_0 \in [\ 0 : \lceil \frac{\rho^{\mathsf{max}}-1}{2} \rceil\ ] \\ \delta^{\mathsf{out}} \cdot \exp(\varepsilon^{\mathsf{out}} \cdot (\rho^{\mathsf{max}} - \rho_0)) & \rho_0 \in [\ \lceil \frac{\rho^{\mathsf{max}}-1}{2} \rceil + 1 : \rho^{\mathsf{max}}\ ] \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The sensitivity of $\rho_0 + \Delta_0$ and $\rho_1 + \Delta_1$ to the execution of a single trade is ± 1. Distribution frz allocates probability mass across multiples of unit trade volume; neighbouring freezing events ρ_t and $\rho_t \pm 1$ are allocated probabilities which differ by factor $\exp(\varepsilon^{\mathsf{out}})$. Since we limit the amount of frozen tokens to the range $[0 : \rho^{\mathsf{max}}]$, we must accept a non-zero δ^{out} probability of violating ($\varepsilon^{\mathsf{out}}$)-correlated-output differential privacy (See Lemma 13).

Parameterization of P_{frz}. The freeze distribution is parameterized by $(\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}}, \rho^{\mathsf{max}})$, but we note that these cannot be chosen independently; parameters are set so the aggregate probability mass of the P_{frz} is 1. We illustrate various parameterizations of $\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}}$ and ρ^{max} in Fig. 3. To achieve $(2.5, 4.5 \cdot 10^{-4})$-correlated-output differential privacy, ρ_{max} must be set to 6, implying that up to 6 unit volumes of each asset type provided by the liquidity provider will be frozen. Lowering the ρ^{max} reduces frozen liquidity, but implies higher privacy parameters δ^{out} or $\varepsilon^{\mathsf{out}}$. For $\rho_{\mathsf{max}} = 6$ and rounds exceeding 10^3 number of submitted orders (as benchmarked in §5.3), we argue the opportunity cost of freezing up to 6 unit volumes of each asset represents an acceptable cost for round-differential-privacy.

Cost of liquidity provisioning. In fuzzy order matching, the worst case liquidity mismatch occurs when all submitted orders are in the equal direction and are all executed or fulfilled. Here, the maximum mismatch in liquidity is exactly the number of clients submitting orders in the round. Thus, the liquidity provider has to provide as much liquidity as number of clients ($x^{\mathsf{liq}}_{0,1} = n$), in addition to ρ_{max} of each asset type in each round. However, in rDP-Volume-match, the exchange rate is decided apriori according to an external reference price; we argue that the vast majority of the liquidity can be sourced directly from the external market trading at the reference price. In the blockchain context, this could be a large Automatic Market Maker with sufficient liquidity, thereby reducing the amount of liquidity required from the liquidity provider to just ρ^{max} of each asset type. We leave the detailed analysis of effective incentivization of liquidity provisioning to future work; we imagine traders submitting trade fees in each round, but do not model this explicitly.

▶ **Definition 10** (**Privacy epoch**). *We define a privacy epoch over the repeated execution of rDP-Volume-match for m rounds, during which the participating liquidity provider contributes amounts of risky and numéraire assets to be frozen in each round; all frozen funds are returned when the m rounds of the privacy epoch are completed.*

We emphasize that the following privacy properties hold for client in- and outputs during the duration of a single private epoch; once the frozen funds are returned, round differential privacy no longer holds. For purposes of mitigating front-running, we argue that the epoch duration should be chosen to be sufficiently long permit the execution of common, long-running honest user strategies. Alternatively, if the frozen funds provided by the liquidity provider are never returned or burnt, the following privacy properties hold absolutely.

We refer to Appendix A of [10] for formal proofs of the following theorem and lemmas which demonstrate round differential privacy for rDP-Volume-matching.

▶ **Theorem 11.** *rDP-Volume-matching is $(\varepsilon^{in} + \varepsilon^{out}, \delta^{out})$-$(\varepsilon^{out}, \delta^{out})$-$m$-round-differentially-private.*

Theorem 11 follows directly from Lemmas 13 and 14, while the latter is demonstrated by leveraging bounds from Lemmas 12 and 13; we refer to the proof strategy in Appendix A of [10].

▶ **Lemma 12.** *rDP-Volume-matching is $(\varepsilon^{in}, 0)$-input differentially private against an adversary that sees the adversarial trade outputs.*

▶ **Lemma 13.** *rDP-Volume-matching is $(\varepsilon^{out}, \delta^{out})$-correlated-output differentially private.*

▶ **Lemma 14.** *rDP-Volume-matching is $(\varepsilon^{in} + \varepsilon^{out}, \delta^{out})$-input differentially private.*

4.2 Round DP double auctions

We propose a round differentially private double auction algorithm, called DP-double-auction for the trusted curator setting. Here, we introduce an initial sub-routine to compute an $(\varepsilon_1^{\mathsf{in}}, 0)$-input differentially-private *clearing price* from trade orders input to the round. Subsequently, rDP-volume-match (§4.1) is performed on the subset of trade orders with price limits consistent with the clearing price.

For each round, we assume a discrete price range $\mathbf{r} = [r_1, ..., r_l]$; without differentially privacy, the discrete price maximizing the number of order matches would be selected; to preserve input differential privacy, the exponential mechanism is applied to determine the clearing price.

Orders in rDP-double-auction. Inputs submitted to DP-double-auction are input in the form of $\mathbf{x} = [(\mathbf{w}_1, \mathsf{dir}_1, \mathsf{id}_1), ..., (\mathbf{w}_n, \mathsf{dir}_n, \mathsf{id}_n)]$, where each valid trade order $(\mathbf{w}_i, \mathsf{dir}_i, \mathsf{id}_i)$, contains bit array $\mathbf{w}_i = [w_{i1}, ..., w_{il}]$, where each bit w_{ij} indicates whether user i is willing to buy $(\mathsf{dir}_i = 0)$ or sell $(\mathsf{dir}_i = 1)$ at price $r_j \in \mathbf{r}$.

Round differentially private clearing price. In the spirit of the round differentially private mechanisms introduced thus far, rDP-double-auction first computes a deterministic clearing price, and then applies a randomized, mechanism to determine the final, $(\varepsilon_1^{\mathsf{in}}, 0)$-input differentially private clearing price. Note that since the clearing price is "publicly" released, there is no private client output privacy to protect. For each discrete price index $j \in [l]$, we aggregate trade orders willing to sell or buy at price $r_j \in \mathbf{r}$. Let S_j denote all sell orders willing to sell at price $r_j \in [\mathbf{r}]$ and B_j denote all buy orders willing to buy at price $r_j \in [\mathbf{r}]$. Then, the number of matched pairs at price r_j is given by $u_j = \min(B_j, S_j)$, resulting in $\mathbf{u_x} = \{u_1, ..., u_j\}$. Here, we interpret $(\mathbf{u_x} : \mathbb{Z}^{\leq l} \to \mathbb{Z}^{\leq n/2})$ as the *utility function* for sampling the final clearing price with the exponential mechanism.

The *exponential mechanism*, first introduced by McSherry and Talwar [18], realizes a probability distribution over a range of events, for which the mechanism designer can express a *utility* score function $\mathbf{u_x}$ applicable to each event; thus, events can be allocated probability mass proportional to $\exp\left(\varepsilon_1^{\mathsf{in}} \cdot \mathbf{u_x}(r)/(2\Delta u)\right)$, where Δu denotes the sensitivity of the utility function to a change to a single input. In other words, the mechanism designer can influence the probability distribution over events by allocating higher utility scores to preferred outcomes.

In DP-double-auction, the sensitivity of $\mathbf{u}_x(j)$ at any discrete price index $j \in [l]$ is simply 1; thus $\Delta u = 1$. A change in an honest input from valid to a dummy order or from a dummy order to valid affects at most one match per price. Therefore,

$$\Delta \mathbf{u} = \max_{\mathbf{x},\mathbf{x}'} \max_{j \in [l]} \mid \mathbf{u_x}(j) - \mathbf{u_{x'}}(j) \mid \leq 1$$

Then, the probability distribution over which the clearing price is sampled is given by the exponential mechanism parameterized by utility function $\mathbf{u_x}$, which in turn is determined from the submitted trade orders $\mathbf{x}$. Thus, the probability of each discrete price $r_j \in \mathbf{r}$ is given by;

$$\Pr[j] = \frac{\exp(\varepsilon_1^{\mathsf{in}} \cdot \mathbf{u_x}(j)/2)}{\sum_{i \in [|\mathbf{r}|]} \exp(\varepsilon_1^{\mathsf{in}} \cdot \mathbf{u_x}(i)/2)} \tag{5}$$

Since the exponential mechanism is $(\varepsilon_1^{\mathsf{in}}, 0)$-input differentially-private over all inputs ([18]), we consume $\varepsilon_1^{\mathsf{in}}$ of our $(\varepsilon^{\mathsf{in}}, 0)$-input differential-privacy budget when outputting the clearing price computed over $\mathbf{x}$, leaving another $\varepsilon_2^{\mathsf{in}}$ for the subsequent rDP-volume-match at price r, such that $\varepsilon^{\mathsf{in}} = \varepsilon_1^{\mathsf{in}} + \varepsilon_2^{\mathsf{in}}$.

rDP-Volume matching at clearing price. We subsequently apply rDP-volume-match from Section 4.1 at sampled clearing price from the preceding exponential mechanism step, returning the trade outputs from DP-volume-match privately to each trading client and frozen liquidity amounts to the liquidity provider.

▶ **Theorem 15.** *rDP-double auction is m-round-differentially-private.*

We refer to Appendix A of [10] for the formal proof of Theorem 15.

5 Round-DP market mechanisms with MPC

To obtain a fair market in practice, we instantiate the trusted curator with a set of MPC parties who execute the market mechanisms described in Sections 4.1 and 4.2 using an MPC protocol. In Sections 5.1 and 5.2, we present a formal description of the proposed market mechanisms, as well as some textual explanation for the steps of the algorithms where some care is required to ensure the efficiency of the MPC execution.

We implement our oblivious algorithms in the online/offline preprocessing paradigm; during a preceding (off-line) preprocessing phase, secret-shared data is generated which is *independent* of secret client inputs. When running experiments, we focus on benchmarking online phases, as pre-processing phase can be outsourced or parallelized. Thus, in Section 5.3, we present online runtimes for both the rDP-volume-match algorithm and the rDP-double-auction algorithm. We show that, even though these algorithms satisfy stronger privacy guarantees than those in previous works on dark pools using MPC, high order throughput is still achieved. Indeed, the runtimes of our rDP-volume-match algorithm are in the same order of magnitude as those of the Bucket Match algorithm from [12], which provides a similar functionality but without round-differential-privacy. The rDP-double-auction algorithm, on the other hand, has a more expensive input correctness check, becoming more expensive as we consider more price points. Even so, it can process around one thousand orders in just 2 seconds when considering 100 price points. Thus, we introduce the possibility for users to choose the prices at which they wish to trade while achieving practical throughput and satisfying round-differential-privacy.

5.1 rDP-volume-match with MPC

The formal description of the rDP-volume-match algorithm instantiated with MPC is presented in Figures 5 and 6. Note that both the order format and the InputCheckVM procedure in Figure 5, as well as the first 2 steps of the MatchVol procedure in Figure 6 are identical to the ones in the Bucket Match algorithm from [12].

Randomness sampling. We note that distributions sampled during the randomized matching response and the liquidity compensation are independent of the input orders, and can thus be obliviously sampled ahead of time by running the procedure NoiseGen in Figure 6 together during the preprocessing phase of the MPC algorithm. The sampling, described in Figure 4, is performed using the inverse transform sampling method, adapted from [16]. To sample a random value from a distribution given by probability mass function P, we start by taking the corresponding cumulative distribution function, $F_X(x) = \sum_{x_i \leq x} P(X = x_i)$. We then sample a secret shared value $\langle z \rangle \in (0, 1]$ uniformly at random, which can be derived from a randomly generated integer as shown in [16]. The desired distribution can now be obtained by taking the first x such that $F_X(x)$ is greater or equal to $\langle z \rangle$.

```
Sample: On input $P$, the probability distribution of a discrete random variable $X$ that may
  take $k$ different values $x_1, ...x_k$:
  1. Sample $\langle z \rangle \in (0, 1]$ uniformly at random.
  2. For all $i$: $F_i \leftarrow \sum_{j=1}^{i} P(X = x_j)$
  3. For all $i$: $\langle c_i \rangle \leftarrow (F_i \geq \langle z \rangle)$.
  4. For all $i$: $c_i \leftarrow \mathsf{Open}(\langle c_i \rangle)$.
  5. Return $x_j$ for the lowest $j$ such that $c_j = 1$.
```

Figure 4 Sampling from a probability distribution P with MPC..

Input format check. Since the orders are secret shared among the MPC parties, a format verification step is required. I.e., we need to check that every order i is such that $(b_i, s_i) \in \{(1, 0), (0, 1), (0, 0)\}$. To do so, we run the InputCheckVM procedure in Figure 5, where orders without the correct format are rejected.

```
InputCheckVM: On input $\mathbf{x}' = [x'_1, ..., x'_n]$, where $x'_i = (\langle b_i \rangle, \langle s_i \rangle, \langle \mathsf{id}_i \rangle)$ and $b_i, s_i, \mathsf{id}_i \in \mathbb{F}_p$:
    Check validity of inputs bits: $(0,0) \vee (0,1) \vee (1,0)$.
  1. Sample $\alpha_i, \beta_i, \gamma_i$ uniformly at random.
  2. $\langle t_i \rangle \leftarrow \alpha_i \cdot (\langle b_i \rangle \cdot \langle b_i \rangle - \langle b_i \rangle) + \beta_i \cdot (\langle s_i \rangle \cdot \langle s_i \rangle - \langle s_i \rangle) + \gamma_i \cdot (\langle b_i \rangle \cdot \langle s_i \rangle)$
  3. $t_i \leftarrow \mathsf{Open}(\langle t_i \rangle)$
  4. If $t_i = 0$ then add $x'_i$ to a list $\mathbf{x}$, otherwise reject $x'_i$.
  5. Return $\mathbf{x}$.
```

Figure 5 Input correctness check for the rDP-volume-match algorithm (from [12], Figure 3)..

```
NoiseGen: Use Sample from Figure 4 to compute the noise for steps 5 and 9:
  - For all $i$: $\langle \pi_i \rangle \leftarrow \mathsf{Sample}(P_{\mathsf{rr}})$ (def. in Eq. 1).
  - $\langle \rho_0 \rangle, \langle \rho_1 \rangle \leftarrow \mathsf{Sample}(P_{\mathsf{frz}})$ (def. in Eq. 4)
rDP-volume-matching: On input $\mathbf{x}'$, $\mathbf{x}^{\mathsf{liq}}$, submitted by $(\mathcal{P}^{\mathsf{trd}}_1, ..., \mathcal{P}^{\mathsf{trd}}_n)$ and $\mathcal{P}^{\mathsf{liq}}$, respectively,
  where $\mathbf{x}' = [x'_1, ..., x'_n]$, $x'_i = (\langle b_i \rangle, \langle s_i \rangle, \langle \mathsf{id}_i \rangle)$, $\mathbf{x}^{\mathsf{liq}} = (\langle x^{\mathsf{liq}}_0 \rangle, \langle x^{\mathsf{liq}}_1 \rangle)$ and $b_i, s_i, \mathsf{id}_i, x^{\mathsf{liq}}_0, x^{\mathsf{liq}}_1 \in \mathbb{F}_p$:
  1. Let $\mathbf{x} \leftarrow \mathsf{InputCheckVM}(\mathbf{x}')$
  2. Let $\mathbf{y}, \mathbf{y}^{\mathsf{liq}} \leftarrow \mathsf{MatchVol}(\mathbf{x}, \mathbf{x}^{\mathsf{liq}})$
  3. Return $\mathbf{y} = [y_1, ..., y_n]$, $\mathbf{y}^{\mathsf{liq}}$ to $(\mathcal{P}^{\mathsf{trd}}_1, ..., \mathcal{P}^{\mathsf{trd}}_n)$ and $\mathcal{P}^{\mathsf{liq}}$, respectively.
Subroutines invoked by rDP-volume-matching
MatchVol: On input $\mathbf{x} = [x_1, ..., x_n]$ and $\mathbf{x}^{\mathsf{liq}} = (\langle x^{\mathsf{liq}}_0 \rangle, \langle x^{\mathsf{liq}}_1 \rangle)$:
    Step [1a] Deterministic matching of buy & sell orders
  1. For all $i$: $\langle B \rangle \leftarrow \langle B \rangle + \langle b_i \rangle$, and $\langle S \rangle \leftarrow \langle S \rangle + \langle s_i \rangle$
  2. Let $\langle c \rangle \leftarrow (\langle S \rangle > \langle B \rangle)$ and $\langle u \rangle \leftarrow \langle c \rangle \cdot \langle B \rangle + (1 - \langle c \rangle) \cdot \langle S \rangle$.
  3. For all $i$: $\langle \mathsf{big}_i \rangle \leftarrow \langle c \rangle \cdot \langle s_i \rangle + (1 - \langle c \rangle) \cdot \langle b_i \rangle$.
  4. For all $i$, let $\langle \sigma^b_i \rangle \leftarrow \sum_{h=1}^{i} \langle b_h \rangle$ and $\langle \sigma^s_i \rangle \leftarrow \sum_{h=1}^{i} \langle s_h \rangle$.
  5. For all $i$, let $\langle \sigma'_i \rangle \leftarrow \langle c \rangle \cdot \langle \sigma^s_i \rangle + (1 - \langle c \rangle) \cdot \langle \sigma^b_i \rangle$
  6. For all $i$, let $\langle \mathsf{match}'_i \rangle \leftarrow (\langle \sigma'_i \rangle \leq \langle u \rangle) \cdot \langle \mathsf{big}_i \rangle$
  7. For all $i$: $\langle \mathsf{match}_i \rangle \leftarrow (1 - \langle c \rangle) \cdot \langle s_i \rangle + \langle c \rangle \cdot \langle b_i \rangle + \langle \mathsf{match}'_i \rangle$
  8. Set $\mathsf{match} = [\langle \mathsf{match}_1 \rangle, ..., \langle \mathsf{match}_n \rangle]$
    Step [1b] Randomized response over order matches
  9. For all $i$:
       - Let $\langle \mathsf{trade}_i \rangle \leftarrow \langle \pi_i \rangle \cdot \langle \mathsf{match}_i \rangle + (1 - \langle \pi_i \rangle) \cdot (1 - \langle \mathsf{match}_i \rangle)$
       - Let $\langle b^{\mathsf{out}}_i \rangle \leftarrow \langle b_i \rangle \cdot \langle \mathsf{trade}_i \rangle$
       - Let $\langle s^{\mathsf{out}}_i \rangle \leftarrow \langle s_i \rangle \cdot \langle \mathsf{trade}_i \rangle$
       - Add $y_i = (\langle b^{\mathsf{out}}_i \rangle, \langle s^{\mathsf{out}}_i \rangle, \langle \mathsf{id}_i \rangle)$ to the output list $\mathbf{y}$.
    Step [2a] Liquidity compensation for sampled trades
  10. For all $i$: $\langle o^b \rangle \leftarrow \langle o^b \rangle + \langle b^{\mathsf{out}}_i \rangle$, and $\langle o^s \rangle \leftarrow \langle o^s \rangle + \langle s^{\mathsf{out}}_i \rangle$
  11. Let $\langle \Delta_0 \rangle \leftarrow (\langle o^s \rangle - \langle o^b \rangle)$ and $\langle \Delta_1 \rangle \leftarrow -\langle \Delta_0 \rangle$
    Step [2b] Randomized liquidity freezing
  12. $\mathbf{y}^{\mathsf{liq}} = (\langle y^{\mathsf{liq}}_0 \rangle, \langle y^{\mathsf{liq}}_1 \rangle) \leftarrow (\langle x^{\mathsf{liq}}_0 \rangle + \langle \Delta_0 \rangle - \langle \rho_0 \rangle, \langle x^{\mathsf{liq}}_1 \rangle + \langle \Delta_1 \rangle - \langle \rho_1 \rangle)$
  13. Update $(\langle \rho^{\mathsf{frz}}_0 \rangle, \langle \rho^{\mathsf{frz}}_1 \rangle) \leftarrow (\langle \rho^{\mathsf{frz}}_0 \rangle, \langle \rho^{\mathsf{frz}}_1 \rangle) + (\langle \rho_0 \rangle, \langle \rho_1 \rangle)$
  14. Return $\mathbf{y} = [y_1, ..., y_n]$, $\mathbf{y}^{\mathsf{liq}}$.
```

Figure 6 rDP-volume-matching algorithm with MPC.

Fuzzy order matching. To achieve the desired differential privacy guarantees, we avoid revealing any of the secret shared values throughout the computation. This is unlike the Bucket Match mechanism from [12], where the trading direction with the most total volume was revealed, and the matching procedure was simplified by opening successful orders as soon as they were matched. As a consequence, we obtain a more complex, oblivious procedure, described in MatchVol in Figure 6. Here, we calculate the cumulative total volume for each i and for each direction, thus obtaining $\langle \sigma_i^b \rangle$ and $\langle \sigma_i^s \rangle$ (note that we need to perform the calculations in both directions to hide which direction has more total volume). We then compare $\langle u \rangle$ (the total matched volume in each direction) with the cumulative volume at each index i, and accept every order i until $\langle u \rangle$ is exceeded. A randomized response over the matches is obtained by using the randomness $\langle \pi_i \rangle$ sampled during MPC pre-processing.

Liquidity compensation. This phase of the oblivious algorithm, realized in step 12 of the MatchVol procedure (Fig. 6), is identical to the liquidity compensation procedure described in Section 4.1, except that we are now operating over secret shared values.

5.2 rDP-double-auction with MPC

The formal description of the rDP-double-auction algorithm instantiated with MPC is presented in Figures 7 and 8.

Input format check. The correctness of the inputs is verified by the procedure InputCheckDA in Figure 7, which checks that $\langle \mathsf{dir}_i \rangle$ as well as every $\langle w_{ij} \rangle$ are bits and rejects order i if that is not the case. For the orders in the correct format, this procedure additionally converts $\langle w_{ij} \rangle$ and $\langle \mathsf{dir}_i \rangle$ into a sequence of pairs $(\langle b_{ij} \rangle, \langle s_{ij} \rangle)$, where $\langle b_{ij} \rangle$ and $\langle s_{ij} \rangle$ represents whether order i is a buy, sell or a dummy for price j (i.e., $\langle b_{ij} \rangle$ and $\langle s_{ij} \rangle$ have the same meaning as in Section 5.1, but are now associated with a specific price r_j).

```
InputCheckDA: On input $\mathbf{x}' = [x'_1, ..., x'_n]$, where $x'_i = (\mathbf{w}_i, \langle \mathsf{dir}_i \rangle, \langle \mathsf{id}_i \rangle)$, $\mathbf{w}_i = [\langle w_{i1} \rangle, ..., \langle w_{il} \rangle]$
  and $w_{ij}, \mathsf{dir}_i, \mathsf{id}_i \in \mathbb{F}_p$:
    Check all inputs are bits.
  1. For all $j$: sample $\alpha_{ij}$ uniformly at random.
  2. Sample $\beta_i$ uniformly at random.
  3. $\langle t_i \rangle \leftarrow \alpha_{i1} \cdot (\langle w_{i1} \rangle \cdot \langle w_{i1} \rangle - \langle w_{i1} \rangle) + ... + \alpha_{il} \cdot (\langle w_{il} \rangle \cdot \langle w_{il} \rangle - \langle w_{il} \rangle)$
  4. $\langle t_i \rangle \leftarrow \langle t_i \rangle + \beta_i \cdot (\langle \mathsf{dir}_i \rangle \cdot \langle \mathsf{dir}_i \rangle - \langle \mathsf{dir}_i \rangle)$
  5. $t_i \leftarrow \mathsf{Open}(\langle t_i \rangle)$
  6. If $t_i \neq 0$ then reject $\mathbf{x}'_i$. Otherwise, continue to the next step.
  7. For all $j$, let $\langle b_{ij} \rangle = \langle w_{ij} \rangle \cdot (1 - \langle \mathsf{dir}_i \rangle)$ and $\langle s_{ij} \rangle = \langle w_{ij} \rangle \cdot \langle \mathsf{dir}_i \rangle$.
  8. Add $x_i = (\langle b_{i1} \rangle, \langle s_{i1} \rangle, ..., \langle b_{il} \rangle, \langle s_{il} \rangle, \langle \mathsf{id}_i \rangle)$ to a list $\mathbf{x}$.
  9. Return $\mathbf{x}$.
```

Figure 7 Input correctness check for rDP-double-auction..

Exponential mechanism. We obliviously determine how many orders can be matched at each price point by calculating $\langle u_j \rangle$ for each price r_j the same way as $\langle u \rangle$ was calculated in the rDP-volume-matching algorithm. The exponential mechanism can now be used to select the best trading price. The probability $\Pr[j]$ associated with price point r_j depends on the corresponding utility value $\langle u_j \rangle$, and since the utility must remain private, the calculated probabilities $\Pr[j]$ will also be secret shared values. While this does not affect the sampling procedure (the algorithm in Figure 4 remains unchanged if the probability mass function is private), computing each $\Pr[j]$ will require the expensive evaluation of a secure exponentiation.

To avoid exponentiation and efficiently compute the selection probabilities with MPC, we use the techniques proposed in [5]. Firstly, instead of considering the selection probabilities as given in Eq. 5, we reduce the complexity by calculating unnormalized probabilities $\langle W_j \rangle$

(called *weights*), where $\langle W_j \rangle = \exp(\varepsilon \cdot \langle u_j \rangle /2)$. The clearing price sampling can later be performed using these weights by multiplying $\langle z \rangle$ with $\sum_{j=1}^{l} \exp(\varepsilon \cdot \langle u_j \rangle /2)$, as shown in steps 4-8 of FindPrice in Figure 8. Secondly, there are two possible solutions for computing the weights according to the value of $\varepsilon_1^{\mathsf{in}}$ (recall that $\varepsilon_1^{\mathsf{in}}$ is the input privacy budget consumed when executing the exponential mechanism; $\varepsilon_1^{\mathsf{in}}$ is public and fixed beforehand):

(i) For $\varepsilon_1^{\mathsf{in}} = 2 \cdot \ln(2)$, we get $\langle W_j \rangle = 2^{\langle u_j \rangle}$. This value can be directly written as $(\langle 0 \rangle, \langle 0 \rangle, \langle 2 \rangle, \langle u_j \rangle)$ by using the floating-point notation introduced in [2]. With this notation, a secret shared floating-point value $\langle f \rangle$ is represented as a tuple $(\langle s \rangle, \langle o \rangle, \langle v \rangle, \langle p \rangle)$ with $f = (1 - 2 \cdot s) \cdot (1 - o) \cdot v \cdot 2^p$, where s is the sign bit (set to 1 when f is negative), o is the zero bit (set to 1 when f is zero), v is the mantissa and p the exponent.

(ii) For $\varepsilon_1^{\mathsf{in}} = 2 \cdot \ln(2)/2^d$, where $d \in \mathbb{N}$, we get $\langle W_j \rangle = 2^{\langle u_j \rangle /2^d} = 2^{\lfloor \langle u_j \rangle /2^d \rfloor} \cdot 2^{(\langle u_j \rangle \mod 2^d)/2^d}$. The weight $\langle W_j \rangle$ can thus be obtained by calculating the exponentiation with base 2 on the integer part of $\langle u_j \rangle /2^d$, and multiplying it by a corrective term $2^{(\langle u_j \rangle \mod 2^d)/2^d}$ which takes one out of 2^d possible values depending on u_j. The 2^d possible terms are publicly pre-computed and the correct one is obliviously selected using $\langle u_j \rangle$.

There is an additional procedure in [5] for calculating the weights for arbitrary values of $\varepsilon_1^{\mathsf{in}}$. This procedure relies on the decomposability of the considered utility function, meaning that clients can locally calculate the weights associated with their own inputs and these can later be combined to obtain a correct global weight using MPC. Since our utility function is not decomposable, this method is not applicable. An alternative for computing the weights for arbitrary $\varepsilon_1^{\mathsf{in}}$ would be to first publicly calculate all the possible weights according to the amount of submitted orders, and then obliviously select the correct weight for each price point. This would however imply several secure comparisons and become inefficient for large amounts of submitted orders and available price points. It is therefore preferable to choose $\varepsilon_1^{\mathsf{in}}$ according to the formats in (i) or (ii), which already provide considerable flexibility.

```
rDP-double-auction: On input $\mathbf{x}'$, $\mathbf{x}^{\mathsf{liq}}$, submitted by $(\mathcal{P}_1^{\mathsf{trd}}, ..., \mathcal{P}_n^{\mathsf{trd}})$ and $\mathcal{P}^{\mathsf{liq}}$, respectively,
  where $\mathbf{x}' = [x'_1, ..., x'_n]$, $x'_i = (\mathbf{w}_i, \langle \mathsf{dir}_i \rangle, \langle \mathsf{id}_i \rangle)$, $\mathbf{w}_i = [\langle w_{i1} \rangle, ..., \langle w_{il} \rangle]$, $\mathbf{x}^{\mathsf{liq}} = (\langle x_0^{\mathsf{liq}} \rangle, \langle x_1^{\mathsf{liq}} \rangle)$ and
  $w_{ij}, \mathsf{dir}_i, \mathsf{id}_i, x_0^{\mathsf{liq}}, x_1^{\mathsf{liq}} \in \mathbb{F}_p$, as well as a list of prices $\mathbf{r} = [r_1, ..., r_l]$:
  1. Let $\mathbf{x} \leftarrow \mathsf{InputCheckDA}(\mathbf{x}')$
  2. $\mathbf{x}^{\mathsf{match}}, \langle c_R \rangle, \langle u_R \rangle \leftarrow \mathsf{FindPrice}(\mathbf{x})$
  3. Execute MatchVol from Figure 6 from step 3 with inputs $\mathbf{x}^{\mathsf{match}} = [x_1^{\mathsf{match}}, ..., x_n^{\mathsf{match}}]$, $\mathbf{x}^{\mathsf{liq}}$,
     $\langle c_R \rangle$ and $\langle u_R \rangle$.

Subroutine invoked by rDP-double-auction

FindPrice: On input $\mathbf{x} = [x_1, ..., x_n]$, where $x_i = (\langle b_{i1} \rangle, \langle s_{i1} \rangle, ..., \langle b_{il} \rangle, \langle s_{il} \rangle, \langle \mathsf{id}_i \rangle)$:
   1. For all $j$: $\langle B_j \rangle \leftarrow \langle B_j \rangle + \langle b_{1j} \rangle + ... + \langle b_{nj} \rangle$, and $\langle S_j \rangle \leftarrow \langle S_j \rangle + \langle s_{1j} \rangle + ... + \langle s_{nj} \rangle$.
   2. For all $j$, let $\langle c_j \rangle \leftarrow (\langle S_j \rangle > \langle B_j \rangle)$ and $\langle u_j \rangle \leftarrow \langle c_j \rangle \cdot \langle B_j \rangle + (1 - \langle c_j \rangle) \cdot \langle S_j \rangle$.
   3. Calculate weights $\langle W_1 \rangle, ..., \langle W_l \rangle$ using Algorithm 3 from [5] on input $\langle u_1 \rangle, ..., \langle u_l \rangle$.
   4. For all $j$: $\langle F_j \rangle \leftarrow \sum_{h=1}^{j} \langle W_h \rangle$
   5. Sample $\langle z' \rangle \in (0, 1]$ uniformly at random and let $\langle z \rangle \leftarrow \langle z' \rangle \cdot \langle F_l \rangle$.
   6. For all $j$: $\langle q_j \rangle \leftarrow (\langle F_j \rangle \geq \langle z \rangle)$.
   7. For all $j$: $q_j \leftarrow \mathsf{Open}(\langle q_j \rangle)$.
   8. $R \leftarrow r_j$ for the lowest $j$ such that $q_j = 1$.
   9. Set $\mathbf{x}_i^{\mathsf{match}} = [\langle b_{iR} \rangle, \langle s_{iR} \rangle, \langle \mathsf{id}_i \rangle]$.
  10. Return $\mathbf{x}^{\mathsf{match}} = [x_1^{\mathsf{match}}, ..., x_n^{\mathsf{match}}]$, $\langle c_R \rangle$ and $\langle u_R \rangle$.
```

Figure 8 rDP-double-auction algorithm with MPC.

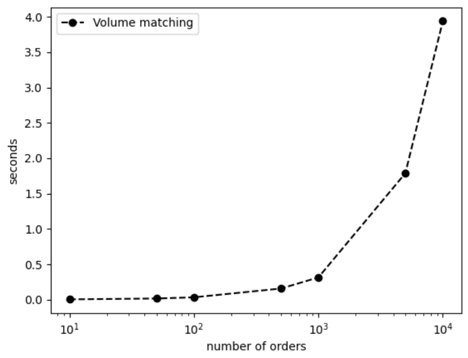


Figure 9 Runtimes in seconds (with logarithmic scale on the x-axis) for the online phase of the rDP-volume-match algorithm.

After a price is selected, we can run the rDP-volume-matching algorithm in Figure 6, starting from step 3 of the MatchVol procedure. Note that since we do not know which orders accept the selected price, every order submitted to the double auction will also be considered when subsequently executing the volume matching. Orders that did not accept the select price will appear as dummies during the matching.

5.3 Experiments

To benchmark the performance of our MPC algorithms, we implemented and executed them using Scale-Mamba [3] with Shamir secret sharing between 3 parties. All the parties are run on identical machines with an Intel i-9900 CPU and 128GB of RAM. The ping time between all the machines is 1.003 ms. Precise numerical values for the results presented here are given in Appendix B of [10].

Online phase of rDP-volume-match. The runtimes for the online phase of one round of the rDP-volume-match algorithm for an increasing number of submitted orders can be found in Figure 9. These runtimes include the InputCheckVM procedure described in Figure 5 which, is identical to the one in the "Bucket Match" dark pool algorithm from [12], and has an average runtime of 0.00013 seconds (0.13 ms) per order. The randomness sampling for the randomized matching response and the frozen liquidity can be done in the preprocessing phase of the MPC, and is thus not considered for the presented results. The runtimes increase approximately linearly with the number of orders (note the logarithmic scale on the horizontal axis), and we see that our algorithm achieves a high order throughput, taking just under 4 seconds to process 10 thousand orders. While InputCheckVM is identical to the input correctness check of the Bucket Match algorithm from [12], our matching procedure MatchVol is slower than the one in Bucket Match. Because of the stronger privacy guarantees we want to satisfy, we cannot reveal intermediary computation results such as which direction has larger total volume or which orders are already matched, as in the Bucket Match. This results in a more complex (and thus more expensive) matching phase. Nonetheless, we note that our runtimes are in the same order of magnitude as the ones presented in [12]. Our rDP-volume-match algorithm can process 10 thousand orders in 3.94 seconds. The Bucket Match algorithm, on the other hand, processes 26838 orders in 4.14 seconds, i.e., it processes around 2.5 times as many orders in a similar amount of time. The slightly lower throughput of rDP-volume-match should still be high enough for most real-world applications, especially considering the improved privacy it provides.

Online phase of rDP-double-auction. The runtimes for the online phase of the rDP-double-auction algorithm for an increasing number of submitted orders and different values of $\varepsilon_1^{\mathsf{in}}$ can be found in Figure 9. These runtimes include the InputCheckDA procedure described in Figure 7, as well as the FindPrice procedure from Figure 8 and the MatchVol from Figure 6 starting from step 3.

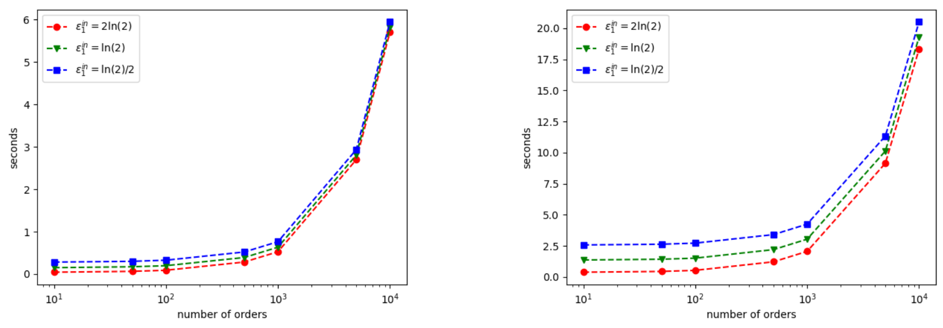


Figure 10 Runtimes in seconds (with logarithmic scale on the x-axis) for the online phase of the rDP-double-auction algorithm with different values of $\varepsilon_1^{\mathsf{in}}$, showing: (left) selection between 10 different price points; (right) selection between 100 different price points. $\varepsilon_1^{\mathsf{in}}$ is the amount of input privacy budget consumed when executing the exponential mechanism to find the clearing price.

The average runtime of InputCheckDA is of 0.00030 seconds (0.30 ms) per order when considering 10 price points and 0.00145 seconds (1.45 ms) per order when considering 100 price points. The percent contribution of this part of the algorithm to the total runtime becomes more significant as the number of orders increases, constituting around 50% of the total runtime across all $\varepsilon_1^{\mathsf{in}}$ values when considering 10 thousand orders with 10 price points, and 70% to 80% when considering 10 thousand orders with 100 price points, depending on the choice of $\varepsilon_1^{\mathsf{in}}$. The FindPrice procedure, on the other hand, does not get significantly slower with the increase in the number of orders. This is also the only part of the algorithm that depends on the choice of $\varepsilon_1^{\mathsf{in}}$, since the method for calculating the weights associated with each price point changes depending on $\varepsilon_1^{\mathsf{in}}$, as described in Section 5.2. As expected, the difference in runtime for different $\varepsilon_1^{\mathsf{in}}$'s becomes more noticeable when considering more price points, with FindPrice taking around 2.2 seconds more with $\varepsilon_1^{\mathsf{in}} = \ln(2)/2$ than with $\varepsilon_1^{\mathsf{in}} = 2\ln(2)$. Nonetheless, this increase remains comparatively small when we consider large numbers of orders.

6 Future work

In this work, we have initiated the study of differential privacy in the trusted curator model, resulting in a definitional framework of round differential privacy, which protects both private inputs and private, yet correlated-outputs. We argue this setting applies to many economic or financial application domains. We introduce round differentially private market mechanisms for traditional finance, but also decentralized finance when instantiated with privacy-preserving smart contracts [4].

We highlight the investigation of *general* correlated-output differentially private mechanisms for common output correlation classes as an interesting avenue for future work. In the setting of standard differential privacy, the Laplace, Gaussian or exponential mechanisms provide “plug-and-play” techniques to transform query algorithms into differentially privacy mechanisms. The investigation of similarly general techniques to achieve correlated-output differential privacy would represent a useful toolkit in the trusted curator setting. Achieving efficiency for such generalized mechanisms with custom MPC protocols would greatly facilitate the deployment of round-differentially-private mechanisms in practice.

References

1 Abbas Acar, Z Berkay Celik, Hidayet Aksu, A Selcuk Uluagac, and Patrick McDaniel. Achieving secure and differentially private computations in multiparty settings. In *2017 IEEE Symposium on Privacy-Aware Computing (PAC)*, pages 49–59. IEEE, 2017. doi:10.1109/PAC.2017.12.

2 Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. Secure computation on floating point numbers. *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA*, 2013.

3 Abdelrahaman Aly, Kelong Cong, Daniele Cozzo, Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Oliver Scherer, Peter Scholl, Nigel P. Smart, Titouan Tanguy, and Tim Wood. SCALE-MAMBA v1.12: Documentation, 2021. URL: https://homes.esat.kuleuven.be/~nsmart/SCALE/Documentation.pdf.

4 Carsten Baum, James Hsin-yu Chiang, Bernardo David, and Tore Kasper Frederiksen. Eagle: Efficient Privacy Preserving Smart Contracts. *Cryptology ePrint Archive*, 2022. URL: https://eprint.iacr.org/2022/1435.

5 Jonas Böhler and Florian Kerschbaum. Secure multi-party computation of differentially private median. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2147–2164. USENIX Association, August 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/boehler.

6 John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. MPC joins the dark side. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 148–159, 2019. doi:10.1145/3321705.3329809.

7 John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. Multi-party computation mechanism for anonymous equity block trading: A secure implementation of turquoise plato uncross. *Intelligent Systems in Accounting, Finance and Management*, 28(4):239–267, 2021. doi:10.1002/isaf.1502.

8 David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri De Ruiter, and Alan T Sherman. cmix: Mixing with minimal real-time asymmetric cryptographic operations. In *Applied Cryptography and Network Security: 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings 15*, pages 557–578. Springer, 2017. doi:10.1007/978-3-319-61204-1_28.

9 David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981. URL: https://www.doi.org/10.1145/358549.358563.

10 James Hsin-yu Chiang, Bernardo David, Mariana Gama, and Christian Janos Lebeda. Correlated-Output Differential Privacy and Applications to Dark Pools. https://eprint.iacr.org/2023/943, 2023. Full paper version.

11 Tarun Chitra, Guillermo Angeris, and Alex Evans. Differential privacy in constant function market makers. *Cryptology ePrint Archive*, 2021. URL: https://eprint.iacr.org/2021/1101.

12 Mariana Botelho da Gama, John Cartlidge, Antigoni Polychroniadou, Nigel P Smart, and Younes Talibi Alaoui. Kicking-the-bucket: Fast privacy-preserving trading using buckets. *Cryptology ePrint Archive*, 2021. To appear at FC’22. https://eprint.iacr.org/2021/1549.

13 Mariana Botelho da Gama, John Cartlidge, Nigel P. Smart, and Younes Talibi Alaoui. All for one and one for all: Fully decentralised privacy-preserving dark pool trading using multi-party computation. Cryptology ePrint Archive, Paper 2022/923, 2022. URL: `https://eprint.iacr.org/2022/923`.

14 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006. `doi:10.1007/11681878_14`.

15 Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014. `doi:10.1561/0400000042`.

16 Fabienne Eigner, Aniket Kate, Matteo Maffei, Francesca Pampaloni, and Ivan Pryvalov. Differentially private data aggregation with optimal utility. In *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC 2014, New Orleans, LA, USA, December 8-12, 2014*, ACSAC '14, pages 316–325, New York, NY, USA, 2014. Association for Computing Machinery. `doi:10.1145/2664243.2664263`.

17 Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1376–1385. JMLR.org, 2015.

18 Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 94–103. IEEE, 2007. `doi:10.1109/FOCS.2007.66`.

19 United States of America before the Securities and Exchange Commission. In the matter of itg inc. and alternet securities, inc., exchange act release no. 75672. `https://www.sec.gov/litigation/admin/2015/33-9887.pdf`, 12 Aug 2015.

20 United States of America before the Securities and Exchange Commission. In the matter of pipeline trading systems llc, et al., exchange act release no. 65609. `https://www.sec.gov/litigation/admin/2011/33-9271.pdf`, 24 Oct 2011.

21 United States of America before the Securities and Exchange Commission. In the matter of liquidnet, inc., exchange act release no. 72339. `https://www.sec.gov/litigation/admin/2014/33-9596.pdf`, 6 Jun 2014.

22 Manas Pathak, Shantanu Rane, and Bhiksha Raj. Multiparty differential privacy via aggregation of locally trained classifiers. *Advances in neural information processing systems*, 23, 2010. URL: `https://proceedings.neurips.cc/paper_files/paper/2010/file/0d0fd7c6e093f7b804fa0150b875b868-Paper.pdf`.

23 Sikha Pentyala, Davis Railsback, Ricardo Maia, Rafael Dowsley, David Melanson, Anderson Nascimento, and Martine De Cock. Training differentially private models with secure multiparty computation. *arXiv preprint arXiv:2202.02625*, 2022. URL: `https://arxiv.org/abs/2202.02625`.

24 Penumbra. ZSwap documentation. `https://protocol.penumbra.zone/main/zswap.html`, 2023.

25 Monica Petrescu and Michael Wedow. Dark pools in european equity markets: emergence, competition and implications. *ECB Occasional Paper*, (193), 2017. `doi:10.2866/555710`.

26 Thomas Steinke. Composition of differential privacy & privacy amplification by subsampling. *CoRR*, abs/2210.00597, 2022.

27 Sameer Wagh, Xi He, Ashwin Machanavajjhala, and Prateek Mittal. Dp-cryptography: marrying differential privacy and cryptography in emerging applications. *Communications of the ACM*, 64(2):84–93, 2021. `doi:10.1145/3418290`.

28 Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965. `doi:10.1080/01621459.1965.10480775`.

SoK: Privacy-Enhancing Technologies in Finance

Carsten Baum
Technical University of Denmark, Lyngby, Denmark

James Hsin-yu Chiang
Aarhus University, Denmark

Bernardo David
IT University of Copenhagen, Denmark

Tore Kasper Frederiksen
Zama, Paris, France

Abstract

Recent years have seen the emergence of practical advanced cryptographic tools that not only protect data privacy and authenticity, but also allow for jointly processing data from different institutions without sacrificing privacy. The ability to do so has enabled implementations of a number of traditional and decentralized financial applications that would have required sacrificing privacy or trusting a third party. The main catalyst of this revolution was the advent of decentralized cryptocurrencies that use public ledgers to register financial transactions, which must be verifiable by any third party, while keeping sensitive data private. Zero Knowledge (ZK) proofs rose to prominence as a solution to this challenge, allowing for the owner of sensitive data (*e.g.* the identities of users involved in an operation) to convince a third party verifier that a certain operation has been correctly executed without revealing said data. It quickly became clear that performing arbitrary computation on private data from multiple sources by means of secure Multiparty Computation (MPC) and related techniques allows for more powerful financial applications, also in traditional finance.

In this SoK, we categorize the main traditional and decentralized financial applications that can benefit from state-of-the-art Privacy-Enhancing Technologies (PETs) and identify design patterns commonly used when applying PETs in the context of these applications. In particular, we consider the following classes of applications: 1. Identity Management, KYC & AML; 2. Markets & Settlement; 3. Legal; and 4. Digital Asset Custody. We examine how ZK proofs, MPC and related PETs have been used to tackle the main security challenges in each of these applications. Moreover, we provide an assessment of the technological readiness of each PET in the context of different financial applications according to the availability of: theoretical feasibility results, preliminary benchmarks (in scientific papers) or benchmarks achieving real-world performance (in commercially deployed solutions). Finally, we propose future applications of PETs as Fintech solutions to currently unsolved issues. While we systematize financial applications of PETs at large, we focus mainly on those applications that require privacy preserving computation on data from multiple parties.

2012 ACM Subject Classification Security and privacy → Cryptography

Keywords and phrases DeFi, Anti-money laundering, MPC, FHE, identity management, PETs

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.12

Related Version *Full Version*: `https://eprint.iacr.org/2023/122`

Funding *Carsten Baum*: Part of the work was carried out while the author was visiting Copenhagen University and supported by Partisia. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of Partisia.
James Hsin-yu Chiang: This work was supported by a DTU Compute scholarship.
Bernardo David: The project was supported by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B, and by DIREC.
Tore Kasper Frederiksen: This work was carried out while working at Protocol Labs and the Alexandra Institute (supported by Copenhagen Fintech as part of as part of the "National Position of Strength programme for Finans & Fintech" funded by the Danish Ministry of Higher Education and Science).

1 Introduction

Modern Cryptography and Traditional Finance. Due to their sensitive nature, financial applications require strong security guarantees. Clearly, it is necessary to ensure authenticity and integrity of any financial operation, *i.e.* guaranteeing that the operation has been ordered by an entity authorized to do so and that this order has not been tampered with. Moreover, it is also necessary to achieve *privacy*, *i.e.* preventing attackers from obtaining sensitive information related to financial operations (*e.g. the identities of entities involved in a transaction and/or the value of that transaction*). In the digital realm, authenticity and privacy guarantees can be achieved against powerful adversaries who control communication networks (*e.g.* the Internet) by means of digital signatures and encryption, respectively.

A Decentralized Conundrum. The meteoric rise of decentralized financial applications based on cryptocurrencies and smart contracts hosted on blockchain platforms brought to light a whole new set of challenges. While traditional financial applications are hosted and executed by financial institutions in a centralized manner, the decentralized nature of blockchain-based applications requires all operations to be verifiable by third parties by means of publicly available records. If only simple cryptographic primitives are employed, this means that sensitive data that was once internally handled by financial institutions must now be exposed on the blockchain in order to perform a financial application. For example, Bitcoin requires revealing the sender and the receiver of a financial token that is transferred, so that a transfer transaction is considered valid if and only if the rightful owner of the token signs it.

Privacy-Enhancing Technologies (PETs) to the Rescue. While sacrificing privacy to achieve decentralization may be acceptable in some situations, most financial operations involving companies and private citizens cannot be conducted in this manner due to a number of reasons (*e.g.* protecting business interests and complying to regulations). In order to solve this issue, the cryptocurrency community turned to Privacy-Enhancing Technologies (PETs) that allow for achieving the same authenticity and privacy guarantees as in traditional centralized financial applications while providing the public verifiability guarantees needed in decentralized blockchain platforms. In particular, many of the first proposals towards this goal involved using a technology called Zero Knowledge (ZK) proof systems [67] : a method that allows the owner of sensitive data to prove a statement about this data without having to reveal it. For example, in the token transfer transaction example, a ZK proof allows a user to prove that an "encrypted" transfer transaction has been signed by the rightful owner of the token, without revealing neither the owner's nor the receiver's identity, *e.g.* as in [15].

From Decentralized to Traditional Finance. The vast usefulness of advanced PETs in blockchain applications also sparked an interest in deploying similar solutions for traditional financial applications. The aforementioned ZK proof technology has also been used in innovative solutions to the Know Your Client (KYC) and Anti Money Laundering (AML) problems commonly encountered in the banking industry, *e.g.*, as in [79, 118, 115, 110]. As in the case of privacy preserving cryptocurrency transactions, in many scenarios an entity wants to prove that they comply with KYC/AML regulations without revealing their identity nor their sensitive data. For example, a client can prove to a third party service provider that their identity has been verified by their bank and that they are authorized to use a certain service and perform operations up to a certain financial volume, while keeping their identity, and other attributes (*e.g.* the list of operations they are allowed to perform) private.

PETs for the Masses - or - From ZK to MPC. The ZK proof technology lends itself extremely well to applications that require a single entity to publicly prove a statement about its private data, *e.g.* the KYC/AML or cryptocurrency examples above. However, it is limited by the fact that the entity who generates a ZK proof must necessarily know all the private information about which the statement is proven. This is a serious limitation in two cases: 1. applications that must process sensitive data provided by multiple entities; 2. applications where certain data (*e.g.* cryptographic secret keys) are far too valuable to be stored on a single device, which leaks the data if its security is compromised. Fortunately, these limitations can be addressed by means of secure Multiparty Computation (MPC) [36, 66], which allows a set of entities to jointly execute an arbitrary program that computes on an "encrypted" version of their private data and only reveals the output of this computation.

For example, specific-purpose MPC protocols have long been used for sealed-bid auctions [70, 25] among entities who do not trust each other, nor a third party auctioneer. In this case, the parties provide as input "encrypted" versions of their bids and jointly compute a program that determines the winner of the auction, without revealing the value of the bids or any other information. In the context of blockchain-based cryptocurrencies, MPC protocols [77] have also been successfully deployed [42] for protecting secret signature keys used to authorize/authenticate transactions. In this case, many entities locally store a "share" of the signing key that does not reveal any information about the key itself unless all shares are united. When a transaction must be signed, all these entities use MPC to jointly execute a program that takes as input all the signing key shares, reconstructs the true key and computes the signature on the given transaction, while only revealing the resulting signature and nothing else. Since knowledge of the key is split among many entities, an attacker now has to compromise many, potentially all, entities instead of a single server.

Systematizing Privacy-enhancing Technologies in Finance

The goal of this SoK is to systematize financial applications that can benefit from PETs, as well as systematizing relevant PETs according to their respective applications and technological readiness. As summarized in Table 1, we consider 5 main classes of financial applications, which are each addressed in the specific section indicated next to the application class name. At a high level, these applications can be potentially facilitated by the PETs indicated in the **PET** column of Table 1, which are introduced in detail in Section 2. In particular, we focus on financial applications that require handling private data from multiple entities, as ZK proof technology for financial applications has been extensively addressed in previous works (*e.g.* [24, 3, 4, 91]). While we aim at providing a general overview of PETs for financial applications covering broad ranges of both PETs and applications, we do not intend to provide an exhaustive review of the PET literature. For each class of applications, we strive to survey the works that introduced the most relevant insights and groundbreaking results, since it would be infeasible to cover every single optimization of each PET that would be relevant for each application.

The financial applications we cover and the respective relevant PETs are summarized as follows:

Identity, KYC & AML (Section 3): Identity management is a classical problem that has the added challenges of Know Your Client (KYC) and Anti Money Laundering (AML) regulations in the financial sector. PETs can be used in these applications to provide robust identity management with privacy preserving methods for enforcing KYC & AML regulations in both decentralized and traditional scenarios.

Table 1 PET stack for financial applications. TSS = Threshold Secret Sharing, DP = Differential Privacy, (F)HE=(Fully) Homomorphic Encryption, PSI = Private Set Intersection, MPC = Multiparty Computation, ZK = Zero Knowledge proofs. See Section 2 for a discussion of each concept.

	PET (§2)
(§3) Identity, KYC & AML	MPC, ZK
(§A, [10]) Legal	MPC, ZK
(§B, [10]) Digital Asset Custody	TSS, MPC
(§4) Markets & Settlement	(F)HE, MPC, ZK
(§5) Future applications	PSI, DP, MPC

Legal Procedures (Section A in [10]): Many legal procedures require evidence to be presented in court. However, in many cases the evidence or even the relevant law/regulation must be kept private. PETs allow for such legal procedures to be conducted without sacrificing neither privacy nor auditability (*i.e.* the ability of any entity to verify that a legal procedure has been properly executed). Furthermore, due to the novelty of PETs it is not always clear how they fit within existing legal frameworks and how they might help and provide utility while fulfilling privacy regulations such as GDPR.

Digital Asset Custody (Section B in [10]): Digital assets such as cryptocurrencies are usually transferred by means of a digital signature, which can only be generated given a secret key. Since storing this key in a single device poses a risk of key leakage, PETs can be employed to distribute the signing power (and thus the power to move the asset) among many entities, in such a way that the system is only compromised if all entities are compromised.

Markets & Settlements (Section 4): Both the traditional and decentralized financial markets use complex trading instruments that may be abused by entities who retain privileged information about trades. PETs provide a robust solution to this issue via distributed "Dark Pools" or privacy preserving DeFi mechanisms (*e.g.* Automated Market Makers) that process trades without revealing any sensitive information to the entities evolved.

Future Applications (Section 5): Besides financial applications that have already been addressed in previous work, we propose that several PETs can be potentially used to address other interesting challenges in finance. In particular, recent advances in PETs enable the execution of advanced machine learning (ML) algorithms on private data, allowing for detecting patterns (*e.g.* for fraud) without revealing neither the ML models nor the data.

2 Available Privacy-enhancing Technologies

Before we describe applications of *Privacy-Enhancing Technologies* (PETs) to finance, we will give a short overview over existing PETs and how mature they are.

Zero-Knowledge proofs. Zero-Knowledge proofs [67] are cryptographic algorithms which allows a prover to convince a distrusting verifier that a certain statement is true. While the statement (usually specified in the form of a program) is known to the verifier, the proof (e.g. a certain input that makes the program output 0) is never leaked to the verifier. There exists a large variety of different ZK proof algorithms, and choosing the optimal proof depends largely on the application. Recently, efforts have been underway to standardize ZK proofs[1] to make them more accessible to practitioners.

Private Set Intersection. Private Set Intersection (PSI) [58] allows two (or more) distrusting parties with respective input sets S_1 and S_2 to securely learn their intersection, i.e. $S_1 \cap S_2$, without revealing the non-intersecting elements to the other party. For example, if party 1 has $S_1 = \{a, b, d\}$ and party 2 has $S_2 = \{b, c, e\}$ then both parties will learn that they have b in common in their sets. At the same time, party 2 will not learn that party 1 also had a, d in its input set and vice-versa for c, e. Highly efficient PSI protocols have been developed recently and some, such as the one developed by Chen *et al.* [39] found applications in industry.

Threshold Secret Sharing. Threshold Secret Sharing (TSS) allows a dealer to distribute [100] a secret x among n different parties, who each receive a *share* of the secret. Given a threshold $t < n$, TSS guarantees that if t or less parties pool their shares together, then they cannot reconstruct any information about x. If instead more than t parties cooperate (*i.e.* pool their shares), then x can be reconstructed. Multiple versions of secret sharing exist, for example with security against share-holders who don't act honestly during the reconstruction of the secret [41, 93]. Moreover, secret-sharing can be generalized so that not a threshold decides about the possibility of reconstruction, but instead any pattern can be used by the sender of the shares.

Multiparty Computation. Cryptographic protocols for Multiparty Computation (MPC) [14, 37, 65] allow 2 or more mutually distrusting parties who each have an input x_i to evaluate an arbitrary function $y = f(x_1, \ldots, x_n)$ on their inputs. MPC guarantees that only the function output y and no other information about the inputs is revealed. One can see PSI as a special case of MPC where the computed function is the intersection of input sets. MPC can be made robust against parties who maliciously deviate from the protocol description, and security usually holds if less than a threshold t of the participants in the computation collaborate to undermine the security. Therefore, MPC can be seen as constructing a *distributed trusted entity.* It is worth highlighting that MPC can be carried out in many different security and communication models. In particular the communication model can have a big impact on the possible use-cases *and* security of the protocol. For example in a *synchronous* model any message sent can be assumed to arrive within a fixed amount of time, whereas in the *asynchronous* model messages can be arbitrarily, even infinitely, delayed. Hence the synchronous model is more suited for protocols being run on a stable network, like a LAN. Recent progress in MPC research has made practical use of MPC possible[2].

Fully-Homomorphic Encryption. Fully-Homomorphic Encryption (FHE) is a special type of encryption scheme first proposed in [95] and later realized in [63]. In FHE, everyone with a so-called public key can encrypt information, while only the holder of the private key can decrypt it later. In addition, given encrypted of data as well as the public key, anyone can *perform computations* on the encrypted data and evaluate algorithms on secret inputs. For example. Given encryption $[x], [y], [z]$ of the values x, y, z, FHE allows to compute an

[1] See `https://zkproof.org/`.
[2] `https://www.mpcalliance.org/`

encryption $[x \cdot y + z]$ of $x \cdot y + z$ or *any other efficiently computable algorithm* on these inputs. The clue is that the decryptor who obtains $[x \cdot y + z]$ will only learn $x \cdot y + z$ but not the *inputs to the computation.* Although concrete FHE schemes are relatively new, the technology is already somewhat mature[3] and powerful testing implementations[4] are available.

Differential Privacy. Differential Privacy [55] (DP) is a technique to compute *add noise to outcomes of algorithms* such that leakage about the inputs of the computation is minimized. The level of the noise is calibrated such that mathematical guarantees about the privacy of the inputs can be given.

Readiness of the technologies. We next try to make a rough evaluation of the market readiness and usability of the different technologies. Our main metric for this is the *Technology Readiness Level* (TRL). Recall that the TRL scale is an estimation of technology maturity, originally developed by NASA and standardized by ISO [71]. The scale goes from 1 to 9, where level 1 indicates the initial studies of moving general research into applied research and level 9, means the system has been proven successful in the real-world. The systems we are considering in this paper are far up the scale so we outline what exactly we mean by the different TRL levels achieved for these systems:

7. Demonstration of the system in an operational environment. E.g. beginning of alpha-testing.
8. The system is fully developed and functioning under expected real-world conditions.
9. The system is actually deployed and used by end-users.

We summarize the main privacy enhancing technologies we consider in this SoK and our judgement of their Technology Readiness Level (TRL) in Table 2.

While the TRL level provides one metric in judging the market application of a technology; overhead (in particular in computation and communication) is another important factor, in particular when it comes to scalability and wideness of deployment.

Hence we have include overhead indicators in Table 2 in order to illustrate the *typical* overhead of the different technologies relate. An interesting observation on overhead is that DP can be applied with barely any computational overhead and TSS (*without* key generation) can in some situations have an overhead only linear in the amount of servers involved. Some ZK on the other hand (in particular SNARKs [17]) has the powerful feature that it can *reduce* the communication and computation overhead over native computation (for *some* of the involved parties only!). This feature has been used widely in practice, in particular in the blockchain setting for example to aggregate validation of multiple transactions through a *zk-rollup* [43].

A note on Trusted Execution Environments. Trusted Execution Environments (TEE) such as Intel's SGX are special modes of modern processors. A processor in its trusted execution setting guarantees that programs and their data are shielded from every other program running on the computer - even the operating system or any user having full access. A secure TEE allows to build many of the aforementioned PETs such as ZK proofs, PSI, MPC etc. "cheaply" and without additional cryptographic tools. In practice, SGX and similar technologies from other vendors[5] are regularly broken and do not offer the protection that they claim. We therefore do not consider it as a PET in this document.

[3] `https://fhe.org/`
[4] `https://www.openfhe.org/`
[5] See e.g. the exhaustive list on `https://sgx.fail/`.

Table 2 Technology Readiness Level (TRL) between 1-9 of different PETs along with how much overhead they *typically* add when used in the *most* suitable contexts. An empty circle means close to no overhead, and a full circle means multiple orders of magnitude. Finally a dot means *less* than native communication/computation. See Section 2 for a discussion of each concept.

PET (§2)	**TRL**	Comp. Overhead	Comm. Overhead	*Demonstrated by*
ZK	9	·	·	Ethereum [43], Filecoin [76]
TSS	9	◔	◔	Zengo [75]
DP	9	○	○	Apple [53]
PSI	8	◔	◔	Apple [16]
MPC	7	◕	◒	JP Morgen [46], Meta [35]
FHE	7	●	◔	Zama [96]

3 Identity, KYC, AML

A general issue facing the financial world is the validation of customer identities and attributes. Laws and regulations require financial institutions, both classical and decentralized, to employ Know Your Customer (KYC) rules, for example to prevent money laundering and to be able to aid in criminal cases - or even to lock accounts in case of sanctions. Being able to correctly determine a legal owner of an account can in itself help in preventing money laundering by precluding the use of fake accounts which could otherwise aid in *smurfing*, see Sec. 3. In the EU this is for example in place through the Anti-Money Laundering Directives and in the US through the Money Laundering Control Act of 1986. In the classical banking setting such validations are carried out through customers going to their physical bank and bringing required documents to prove their identity, residence or perhaps even criminal history, of which the bank would keep a copy. However, with the advent of online-only banks such as Lunar, Revolut and N26, along with crypto-currency exchanges like Binance and Coinbase, such validations become tricky, as there are no physical locations to validate identities.

Today KYC is instead carried out online, and in many cases through machine learning algorithms, where customers upload copies of their data which gets validated. When it comes to security this unfortunately has several disadvantages: i) it is easy to create a picture of a document or manufacture it, or even modify some data of a real document [105]; and ii) the leakage of legitimate documents online allows an adversary to steal identities. Simply considering how often a copy of ones passport is needed (e.g. basically any hotel or accommodation in any country), it is not hard to see that copies of legitimate documents will be easy to find on the dark market. Even though requirements can be made to include selfies or short videos to validate authenticity, this has turned into a race against Photoshop and deep fakes, which have shown tremendous advancement in the recent years [106]. While such attacks are also possible in physical space (i.e. creating fake documents and having them validated by a human), they are significantly more cumbersome due to human involvement and more expensive to mount, and therefore do not *scale* like digital-only attacks. Thus it is clear that the digital attack vector on KYC is the weakest link in account validation.

Identity management

One possible way of combating attacks when validating digital copies of physical identity documents is simply to move the documents into the digital realm. Digital signatures and revocation systems can ensure that digital documents are legitimate. This is done by combining them with an identification scheme where a user needs to prove they know a password/key used in the construction of their digital identity. This can prevent theft by simply copying the digital document. Often a simple password or key is not deemed secure enough in financial applications by law [90], and a second factor is required.Thus the use of authentication apps is common in electronic ID (eID) solutions, like the Danish MitID. Such eID solutions validate user-identities by a trusted issuer during setup, allowing other applications to piggy-back on existing validation. This naturally comes with a risk of compromise or identity sharing through the eID provider, although it may arguably be harder than with their physical counterparts.

Single Sign-On. eIDs are typically validated by a centralized and trusted server that is able to perform relevant logging, and hence poses a risk to user privacy. Furthermore, such identity management is not exclusive to official or government identities, but can involve any kind of self-reported identity, which is the case for example for a Facebook or Google account. These platforms act as *federated identity management* services, allowing the sharing of the user's identity, along appropriate attributes of the user, to third-party websites. Thus facilitating a *single sign-on* (SSO) system. In this setting, the server validating the user's identity is known as the *identity provider* (IdP), which would be Facebook or Google in the above example. The third-party website is known as the *service provider*. This could for example be Netflix or Spotify. The idea of an SSO service goes beyond simply having an IdP facilitating a user authenticating towards a service provider. In fact an IdP may gather certified attributes about a user from multiple trusted issuers, and sign off on the user indeed being validated to have such attributes. This for example happens when Facebook validates that a given user has access to a specific email account, or phone number. While using an SSO makes things much simpler for a user, it also is a big privacy issue as an IdP now holds a large amount of the user's personal information, along with knowledge of whenever the user users this information and towards which service provider. Furthermore, it also means that large amount of trust has to be put in the IdP as they would be able to impersonate any of their users towards any service provider. While such a thing is also possible for any attribute issuer (to a lesser extent) it becomes more of a problem for an IdP as they must be user-friendly enough that they can be used several times a day and since their only job is authentication. Hence becoming more exposed. Beyond this, simply using an SSO service can also lead to *traceability* and *linkability* of the user across the web. Traceability means that a user can be identified from the data resulting from using their eID. Whereas linkability means that it is possible for different service providers to find out if they have the same users. This can be an issue even if the user is authenticated using a pseudonym, since all it takes is one sharing of personal data, such as credit card information at a service provider, to de-anonymize the user. However, works like PASTA [2] and PESTO [12] use threshold cryptography to enhance the security of IdPs and limit traceability and linkability without reducing the usability. I.e. password based authentication can still be used and they remain compliant with solutions like OAuth and OpenID Connect. While they only focus on password-based authentication, they can be generalized to support multi-factor authentication [57] and thus be used when multi-factor authentication is required for financial compliance, as e.g. in Europe according to PSD2 [90].

Decentralized Identifiers. With the advent of blockchain technologies a lot of work has sprung up, trying to remove centralization from the management of eIDs. This is generally known as a Decentralized Identifier (DID) [102], but often also called Self-Sovereign Identity

(SSI), specifically when the user is in full control over the usage of their decentrally stored eIDs. The overall idea is that any kind of attribute provider issues a pseudonym to a user's blockchain account, reflecting a specific attribute. The user can then later use the pseudonym to prove certain certain attributes, or to simply get a reusable link to their pseudonymous identity at the same identity provider. However, it is clear to see that this basic construction is unfortunately not enough to ensure privacy, as again it possible to link the user across the internet (or blockchain) through their pseudonym. For this reason DID systems are starting to incorporate more advanced cryptographic constructions allowing users to anonymously prove that they hold a certain pseudonym towards a service provider (in order to facilitate authentication). Such a construction is known as a cryptographic *credential*.

Camenisch and Lysyanskaya [29] were the first to show a fully self-managed solution allowing users to prove their identity has been certified by a trusted provider, in an anonymous manner. Their credential construction affords validation of issuance from a trusted authority, while allowing the user to anonymously use it and preventing anyone who does not know the user's key[6] to impersonate it. However their construction did not allow the validation of arbitrary predicates on the attributes. Something which is needed in many financial situations. Consider for example the case for loan or insurance issuance, where the customer's financial situation or health status has to be validated. Classically these must be provided as signed physical documents from the customer's attribute provider (such as credit bureaus), but the line of work on credentials, known as *attribute based credentials* shows how to achieve this in the digital sphere [30, 98] with cryptographic security and privacy guarantees. It was later shown how to compute arbitrary predicates on the certificated attributes [27]. Further development of such schemes into commercial products have been done by both IBM with their Idemix framework [31] and by Microsoft through U-Prove [88]. The underlying primitives have even been taken up by standardization frameworks such as W3C [101]. Still, despite such commercial traction, widespread adoption is still lacking. Moreover, in the context of DeFi systems, decentralized versions of anonymous credentials [62, 26, 5, 48] have been proposed.

One could imagine that the requirement for self-managed private keys could be the reason that such approaches lack adoption since the regular news bulletins of people having lost their cryptocurrency keys, show that self-administered key management is not for the general public. However, multiple solutions based on threshold cryptography can be used to securely store keys under a client's password [28, 72]. A more likely explanation might be the need of existing attribute providers to completely change their work-flow and systems, without any direct financial, legal or customer requirements.

Deploying Privacy Preserving Identity Management. Fortunately, recent research have shown how PETs can be used to get certified attributed from issuers without modifying existing infrastructure, when such attributes can be retrieved from the provider through TLS-secured connections. The Town Crier system [117] shows how to construct certified attributes using secure hardware (like Intel SGX and using a TLS connection with a trusted provider). Concretely, they discussed how such certified attributes could be relayed to smart-contracts to allow more advanced decentralized user-attribute validation. Later, DECO [118] then showed how to remove the need for secure hardware and replace it with MPC while achieving the same goal. However, they extended their construction to also integrate with

[6] Allowing the user to fully control the use of their credential through a single key can be conceptually advantageous.

zero-knowledge proofs, allowing clients to construct certified proofs of arbitrary *predicates* on attributes from any provider, trusted through a TLS certificate which provides online access to the user's attributes. This could for example include a bank providing online banking access, where a user would then be able to construct a proof that they hold a bank account with e.g. at least $20.000. If the user's government provides an online residency portal, then it could also be used for the users to prove that they legally reside in a given city in a given country without leaking their exact address.

The CanDID system [79] fully realizes a DID system with legacy support through either Town Crier or DECO. This is achieved through the usage of an MPC committee that validates legacy identity data and constructs a zero-knowledge friendly credential. Based on this credential, a user can prove arbitrary predicates on their attributes towards any provider.

Using attribute based credential allows the construction of fully private identity and attribute-based systems. However, in some situations full privacy might be undesirable, we would rather want to privately validate whether transactions are permissible based on attributes or identity, for example by ensuring that the identity of the credential holder is not on a deny-list. Kohlweiss *et al.* [74] showed that such a system can efficiently be constructed on top of credentials. The construction allows an auditor to specify any predicate on the attributes in a credential, where the identity of the credential holder gets leaked if the predicate is fulfilled. Such conditional privacy leakage could prove tremendously helpful in fighting money laundering as we discuss next.

Anti Money Laundering. Money laundering is the process of concealing the origins of money, such as financial gains from drug trafficking or other serious crimes, by changing its origin to a benevolent source. This is because criminals must acquire many services and goods in the regular economy: put simply, most luxury car dealers don't accept briefcases full of bills. Money laundering is a huge problem in the financial sector: the estimated amount of laundered money is at the level of 2-3% of the national GDP in the US alone, excluding tax evasion [94, Chap. 2].

Getting large amount of illegitimate cash into the financial system requires multiple steps and multiple accounts to avoid raising suspicion. Simply getting dirty money into the system is known as *placement*. A concrete and common approach for this is known as *smurfing*, where multiple legal people deposit small amounts of money for a criminal, with the promise of earning a small amount as a kick-back. After a period of time the smurfs move the money out of their accounts (minus their fee), by transferring to other accounts controlled by the criminal. If this process is done with small amounts, and the receiving party's account is not flagged, then smurfing is hard to identify[7].

Once the money is in the legal financial system, it needs to be mixed with legitimate transfers, to counter suspicions caused by the initial transfers. This involves creating reasonable and justifiable transfers among multiple accounts of multiple entities in a process known as *layering*. By setting up a layering scheme through multiple banks, in different legal jurisdictions, using different legal entities, it becomes almost impossible to trace the flow of money. This is because the involved banks are (reasonably!) not allowed to communicate private account and customer information about the sender and recipient of a money transfer. After the layering, the money is finally moved out of financial institutions and into legitimate investments such as real estate or legitimate businesses. This last step is known as *integration*.

[7] This step is sometimes also realized through other means, such as deposits from cash-driven businesses such as laundromats or food trucks.

What banks do to counter money laundering. As banks cannot share account and customer information with each other it is extremely hard for them to trace dirty money during layering. To address this, banks use multiple approaches usually subsumed as Anti-Money Laundering (AML) techniques. For example, banks internally use a *suspiciousness* score for customers. It is based on a *base* score, which is derived from the meta information about the account and its owner. The score may be derived from e.g. age of the account/holder, amount of money in the account, expected income and nationality of the owner. Through transfers, the base score is then updated, e.g. based on the score of the account a transfer goes to or comes from if both sender and receiver account are held by the same bank. If they instead are held by different banks, then metadata such as the amount of money going in/out and the frequency of the transfers is used in updates.

Finally, banks do have one common tool in measuring the suspiciousness of transfers, and that is a common, yet secret, grey list. This grey list contains accounts that have been deemed significantly suspicious, but for whom no provable money laundering has been identified yet. A transfer to or from a grey-listed account significantly increases the suspiciousness score. At certain time intervals, the suspiciousness score of an account is checked against a certain threshold and if the score is too high, then it gets flagged for manual[8] inspection.

What can banks do? Due to GPDR and other privacy laws, it is not possible for banks to directly share meta-information about accounts or its owner without their consent. Furthermore, if a bank finds a flagged account it believes is engaging in illegal activities then when informing authorities, it must be able to *explain* to said authorities how they came to this conclusion. Hence the bank's judgements must be auditable by a third party. If the conclusion depends on data received from other financial institutions, the bank must be able to point to this data and the third party must trust it as well. While data from banks from within the same legal framework (the EU, USA, etc.) is usually considered as valid, data from international banks, in particular those from countries with a history of corruption, has less trustworthiness.

Implementing sufficient and efficient AML techniques is also difficult due to the quantities of information involved. AML technologies should ideally be scalable to include all transactions and accounts. At the same time, even a limited AML technology which only covers cross-country transfers or an arbitrary subset of accounts, could still make a substantial dent into the suspected large amount of money laundering currently going unnoticed.

Cryptography and AML. The conjunction of AML and MPC is new and the main bodies of work on the topic are by Zand *et al.* [115] and Egmond *et al.* [110]. Zand *et al.* show how computation on secret data can be used to notify an auditor of suspicious behavior. Egmond *et al.* show, in collaboration with multiple banks, how to use additively homomorphic encryption to obliviously update risk scores, and eventually (with consent from collaborating banks) decrypt the risk scores and flag accounts and customers appropriately.

However, related to this is the area of auditability of confidential transactions. As for example discussed by Tomescu *et al.* [108] where users are given a limited monthly "anonymity" budget. This budget is a certain amount of currency they are able to transfer anonymously per month. However, transfers surpassing this amount, is subject to deanonymization and clearance by a trusted auditor.

[8] In practice it turns out that about 95% of automatically flagged accounts are false-positives.

Finally, we note that a survey of real world concepts using PETs to combat financial crime has been conducted by the Future of Financial Intelligence Sharing consortium [81]. Unfortunately, many of their mentioned solutions require a trusted party to be involved.

As mentioned above, AML in centralized banking is challenging as the transaction graph is hidden due to e.g. privacy regulations. However in the decentralized finance space, such transaction graphs are usually visible. This is why most popular cryptocurrencies, such as Bitcoin, Ethereum or Cardano, are only pseudonymous and not anonymous[9]. Cryptocurrency exchanges such as Coinbase and Binance allow to turn large amount of cryptocurrency into Fiat currencies. These exchanges are required by law to enforce know-your-customer (KYC) rules. Through the help of transaction graph analysis firms such as Chainalysis, it has become hard to launder money using pseudonymous cryptocurrencies.

Researchers have also proposed mechanisms to enforce AML even if transactions are kept private. This includes using an escrow system where anonymity and privacy can be broken in case suspicious activities occur, such as transfers to or from an account known to be used by criminals [97, 86, 8, 48]. Such escrow mechanisms does not necessarily imply the usage of a trusted third party, as the data for escrow activities can e.g. be shared using Threshold Secret Sharing. Another approach is to specify a small budget per client which they can use every month for anonymous payments. After the client has made more transactions than covered by this budget, any future transactions can be traced [113, 108]. Although seemingly a good compromise between privacy and security, this does still pose a risk to smurfing.

4 Markets & Transaction Settlement

In this section, we systematize PETs in market and settlement applications.

In financial markets, there is a need for auctions and markets with *fairness* guarantees, as rational actors are incentivized to collude and front-run honest parties, if the true valuation or trade-intent of the latter is revealed. Here, we first consider the *traditional finance* setting (§4.1), where the settlement of transactions is handled by traditional, asynchronous settlement processes. In the presence of a *public ledger* (§4.2), settlements occur *synchronously* and *immediately* after a transaction is completed. Such a mechanism also permits the "netting" of inter-bank payments (§4.3) to minimize the liquidity requirements on participating banks; this must done with PET approaches, since the public ledger would otherwise expose all individual payment orders, a clear breach of consumer privacy.

Finally, we highlight approaches to achieve bidder privacy in *demand-response* electricity markets (Appendix C in [10]), which coordinate the remote scheduling of power consuming devices to match forecast production from sustainable production sources; the submission of granular device-level information to an auction in the clear can reveal the activity and presence of customers at home, violating their privacy.

4.1 Markets in Traditional Finance

The first setting reflects an idealized view of *traditional finance*, where accounts and balances are generally maintained by financial institutions and considered private. Here, the settlement of auctions, or exchange transactions, occur asynchronously; whilst the *counterparty risk* from

[9] We note that there exist privacy-focused blockchains like ZCash, Monero, or Dash that hide the transaction graph. Moreover, one can build private transactions on top of non-privacy focused blockchains using e.g. Zether [22] that leverages encryption and ZK proofs. Finally, mixers such as Tornado [92] take transfers from many users and put them into a holding account, from which they can later be transferred to the intended recipient.

Table 3 Auctions & Markets: no benchmarks (○), preliminary benchmarks (◐), benchmarks achieving real-world performance with traditional market parameters (●).

Setting	**Applications**		**Privacy**	**Benchmarks**	**PET**	**Works**
Markets in Traditional Finance (§4.1)	Distributed sealed-bid auctions	Single-sided	Bid privacy	○	MPC	[56], [70], [25], [85]
		Double-sided	Bid privacy	◐	MPC	[19], [18]
		Public verifiability	-	○	HE+ZK	[89]
	Distributed Dark Pools	Continuous matching	Order privacy	◐	MPC	[33]
		Periodic matching	Order privacy	●	MPC	[34], [44]
			Order privacy	◐	FHE	[7]
		with many assets	Order privacy	●	MPC+HE	[34]
		with many servers	Order privacy	●	MPC	[44]
		Public verifiability	-	○	HE+ZK	[107]
Markets on Public Ledgers (§4.2)	Decentralized sealed-bid auctions	Single-sided	Order privacy	○	MPC+ZK	[6], [49], [60]
	Privacy-preserving Decentralized Exchanges	Futures Periodic matching	Net position privacy	◐	MPC+ZK	[80]
		Periodic matching	Balance privacy Partial order privacy	◐	MPC+ZK	[68]
			Order privacy (Balance privacy)	◐	MPC+ZK	[11], ([9])
		Intent-based order matching	Balance privacy Order privacy	◐	ZK	[20], [114]
			Balance privacy Order privacy	●	WKA	[87]
Settlement on Public Ledgers (§4.3)	Liquidity preserving inter-bank netting	-	Payment privacy	◐	ZK	[32]
			Payment privacy & Robustness	◐	MPC+ZK	[50]
Demand-Response Markets (Appendix C in [10])	Distributed auctions for demand flexibility	Double-sided (Single buyer)	Device power constraint privacy	◐	MPC	[1], [119] [59]

defaulting on obligations implied by pending transactions is real, we consider it an orthogonal challenge addressed in the public ledger setting (§4.2, §4.3). Clearing prices and executed volumes are considered public information as this information is forwarded to institutions executing the settlement. This first setting intends to achieve resilience against dishonest venue operators and participants attempting to obtain a financial gain from unwarranted information flow. Communication between parties generally assumes direct, authenticated channels, implying the knowledge of identities and a pubic key infrastructure.

Distributed Sealed-bid Auctions. One-off, sealed-bid auctions are frequently performed in the sale of frequency-spectrum rights, government contracts, real-estate and other private items, such as art. In open-cry, single-sided auctions, bids are broadcast publicly until no additional bids are made. However, the leakage of bids or orders can be exploited by the adversary for financial gain. In Vickrey auctions, where the winner pays the price submitted by the second-highest bid, the auction operator collecting the submitted bids is incentivized to collude with other bidders to increase the second-highest bid price and maximize auction fees. The auction operator must also be *trusted* to not reveal anything about submitted bids, in order for all bidders to submit their *true valuation.* The advent of the public, commercial internet coincides with the first protocol proposals which permit the execution of one-time, sealed-bid auctions by distributing the role of the auction operator, thereby removing the need for a trusted auction venue.

Franklin et al. [56] propose a one-sided auction protocol, where the auction venue is distributed amongst multiple servers; bidders submit their signed bids as *verifiable secret-shares* (VSS) to participating servers during the bidding phase. Subsequently, bids and signatures are jointly reconstructed by all servers, upon which all bid information becomes public. Verifiable secret-sharing ensures that bidders submit well-formed bids. As long as a single server is honest, the reconstruction of bids will not occur before the end of the bidding phase. However, it is often important to protect the privacy of bids, even if they are not successful; the valuation may reveal a bidding strategy for another, related auction.

In the work of Harkavy *et al.* [70], MPC is deployed to maintain the privacy of all submitted bids; only the winning bid is made public. Naor *et al.* [85] propose a variant of MPC with garbled circuits, to reduce the rounds of communication, thereby improving performance. Cachin [25] builds a purpose-built, privacy-preserving protocol, which permits the comparison ($>$) of *prices* between two parties with the help of an untrusted third party. An auction determining the highest bidder is then constructed from this primitive.

The first work to demonstrate the feasibility of privacy-preserving (one-time) *double-sided, sealed-bid auctions* was proposed by Bogetoft *et al.* [19]. Later secure auctions were used in practice [18], to facilitate the auctioning of sugar beet delivery contracts in Denmark. Concretely, farmers producing sugar beets hold contracts which represent an obligation and right to deliver beets to the the (single) Danish sugar beet processor Danisco. The trading of such contracts permits the reallocation of contracts to the most efficient producers, but such an exchange run by Danisco would permit it to learn information about the economic circumstances of producers, potentially compromising sugar beet farmers during contract negotiations. The matching and determination of a price computation from 1229 buy and sell orders was achieved in approximately half an hour by an MPC committee of 3 servers; a throughput volume sufficient for a one-time auction, but unacceptable for traditional electronic security exchanges. Notice, however, that cryptographic techniques have improved drastically since this work. Such an auction would run much more efficiently today.

Publicly verifiable auction operators are proposed in the work of Parkes *et al.* [89], a weaker alternative to implementing the auction operator with MPC; instead, the dark pool venue is still operated by single entity, but provides cryptographic proofs that the auction algorithm is performed correctly by the venue operator. Whilst this prevents the auction venue from *manipulating* the correct evaluation of auction bids, it does not prevent the auction operator from leaking bid information to malicious participants.

Distributed Dark Pools. Dark pools have gained adoption in traditional finance as venues where submitted orders are not publicly accessible, thus minimizing the potential price impact caused by signalling *trade intent* to front-running market participants. As is the case with auction venues, the dark pool operator must be trusted to not *share* the order flow information with malicious participants, a trust assumption that is frequently violated in practice (Table 1 in [34]), motivating the need for distributing the role of the venue operator.

Whilst the matching of orders in a *secret* order book is a natural domain of MPC, we observe that current proposals illustrate specific configurations and architectures for distributed dark pools that may or may not match throughput observed in traditional, centralized dark pool markets. In particular, the choice of auction clearing algorithm remains a deciding feasibility factor. Whilst secret-sharing based MPC schemes permit secret computation with n participating servers, the runtime bottleneck generally lies in the amount of *communication* that is required between servers. This is because MPC has a specific model of defining computations, and certain auction clearing algorithms can be realized with less communication overhead in MPC than others.

In the case of auction clearing algorithms, which must compute a *clearing price* from current bids and sell orders, the *sorting* thereof by *price limit* induces many *comparisons* $(>, <, =)$ between secret-shared values, which in turn imply sub-protocols generating the majority of communication cost. Alternatively, order matching based on volume only (where prices are determined by third party price feeds) can greatly accelerate throughput, as the expensive clearing price evaluation is not required. Furthermore, whether orders are processed *continuously* or *periodically* also greatly affects the real-world applicability of the following distributed dark pool protocols.

Continuous Double Auctions with MPC: A recent line of work by Smart *et al.* [33, 34, 44, 45] implements and examines real-world, double-side auction algorithms. In the initial work [33], continuous double auctions (CDA) are implemented in a distributed fashion across servers running an MPC. Continuous double auctions maintain a limit order book (LOB) where buy and sell orders are ordered by ascending and descending price respectively; *each incoming order* is matched against one or more LOB orders if it crosses the "spread" between best buy (or sell) prices; its remaining volume is then inserted into the LOB. It is also the most *expensive* exchange algorithm since (1) each single order must be matched against m other fulfilled orders and (2) its remaining trade volume must be inserted into a (potentially large) order book of N size. Benchmarking such an algorithm requires specifying the *expected* state of the order book, given the sensitivity of CDA run-time on (1) the average number of matched orders m; and (2) the expected order book length N. In the dark CDA implementation of Cartlidge *et al.* [33], run with 3 servers and Shamir-sharing based MPC, a worst-case throughput of $34 - 43$ orders per second for LOB parameters $m = 3$ and $N \approx 30$ is achieved. This work demonstrates that distributed CDA with MPC cannot yet match the throughput volumes of traditional CDA venues[10]. In contrast, *periodic* order matching greatly improves the performance of distributed dark pools.

Periodic Double Auctions with MPC: Periodic auctions in the dark pool setting implemented with MPC promise throughput that match those of traditional dark pool markets, as shown in these works by Cartlidge *et al.* [33, 34]. In periodic auctions, limit orders are submitted during an open auction period after which a clearing price is computed during the clearing phase, which maximizes the volume of matched orders (unmatched orders are carried over to the next round). In contrast to CDA algorithms, where orders are processed individually against a potentially large order book, periodic auctions only need to compute a single clearing price for the entire batch in a given period. In fact, real-world order execution throughput has been achieved with a realistic number of asset pairs.

In [34], the London Stock Exchange Group's *Turquoise Plato Uncross*, a widely-used traditional dark pool supporting thousands of assets, is implemented with promising results. Based on volume observed on the real-world Turquoise Plato Uncross venue, it is assumed that order book clearing occurs at most every 5 seconds, where at most 2000 newly input orders must be processed across an asset universe of 4000 financial instruments. This throughput was successfully handled by smaller MPC committee sizes of 2 (dishonest majority) and 3 (honest-majority), but required multiple MPC instances, each handling orders trading a small subset of all assets (Figure 1). For example, ~ 280 MPC committee instances are each randomly assigned 16 assets in each round by a *gateway* engine. This gateway periodically

[10] In their work, the runtime of MPC pre-processing is neglected, which represents a non-trivial "hidden" computational cost that can be performed during "offline" hours or outsourced to dedicated pre-processing workers; pre-processing generally does not limit the maximum, sustainable throughput of MPC.

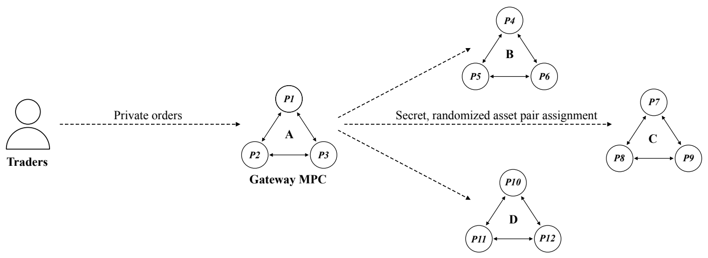


Figure 1 In [34], a gateway MPC (A) distributes inbound received orders across multiple MPC committees (B,C,D) to improve order clearing throughput. MPC servers never learn the asset pairs its committee is assigned in each round.

reassigns asset subsets to new MPC instances in order to break potential linkages between orders across time periods. We note that auction algorithms are not entirely *oblivious.* Indeed, the direction of orders are leaked in [33, 34], whilst volumes and order limits remain private.

Complementary follow-up work by Da Gama *et al.* [44, 45] both focus on (single-asset) privacy-preserving *volume matching*, which enables further performance gains since the clearing prices are determined by an external reference price; [44] introduces an improved MPC volume matching algorithm which permits dummy orders and hides the trade direction. [45] scales volume matching up to MPC instances consisting of $\sim$ 100 servers and shows the economic costs associated with operating such a single server in a MPC instance of up to 100 servers to be below $\sim$ 0.10 USD and $\sim$ 0.025 USD for computation and network communication respectively in each auction round; the negligible cost demonstrates the feasibility of *market participants* contributing to the distributed operation of dark pools.

Periodic Double Auctions with FHE: JP Morgan has demonstrated initial results in work by Balch *et al.* [7] to realize dark pool venues where the venue operator is not distributed, but computes the periodic volume matching over data encrypted under a jointly controlled public key; here, the secret encryption key material used in the *threshold fully homomorphic encryption* is held in secret-shared form by all participants (Figure 2). While the computation can be done by one party on the ciphertexts (not knowing their plain values), the participants then later take part in a so-called distributed decryption protocol which reconstructs the outcomes to the venue operator. Whilst [7] benchmark periodic volume matching implemented with threshold FHE, the omission of the *partial decryption sub-protocol* complicates the evaluation of its performance. Still, FHE offers an alternative approach to secret sharing-based MPC which promises competitive performance; fewer communication rounds are required, since computation is performed locally by the dedicated venue operator over encrypted data, although local computation (over encrypted data) is more costly.

Publicly, verifiable dark pool operator: Similar to verifiable one-sided auctions [89], a weaker notion of order privacy for dark pools is proposed in the following work by Thorpe *et al.* [107], where the venue operator only reveals a homomorphically encrypted order book to traders; the operator itself, however, maintains the encryption key and can thus compute over the order book plaintext. Each update to the public, encrypted order book triggered by a submitted order is accompanied with a zero-knowledge range proof generated by the operator; any public party can locally re-compute the claimed update over encrypted values and verify zero-knowledge range proofs that guarantee that the plaintext values lies within certain ranges, thereby enabling verification of comparison statements between encrypted values. This system ensures the correctness of each order book update without revealing the order details themselves. [107] implements the CDA algorithm in a publicly verifiable manner; as in [89], this approach does not prevent any misuse of the order information held by the operator.

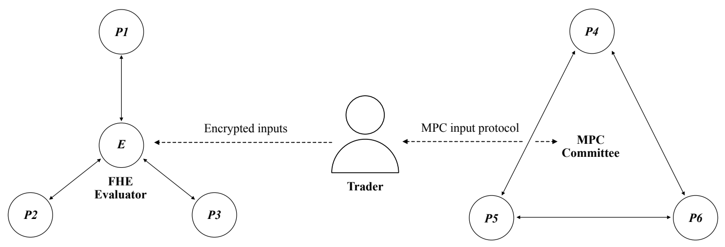


Figure 2 In [7], market auctions are implemented with fully homomorphic encryption (FHE); here, the encryption key is jointly generated by key servers (P1-P3), but the FHE evaluation is solely performed by the evaluator, who never learns the plaintext of the inputs or intermediary results. Decryption of the FHE output requires interaction with all key servers. In contrast, private computation with MPC in [33] requires interaction amongst MPC servers (P4-P6) for each (multiplicative) operation.

4.2 Markets on Public Ledgers

The advent of public ledger protocols [61, 73, 64] resulting from permissionless participation of servers across the public internet promises a truly "server-less" system of transaction settlements, no longer dependent on any single trusted intermediary. The state of the ledger is public and its integrity is publicly verifiable (by any online party) through local verification of all previously finalized transactions sequenced in form of an append-only list, or *blockchain*. Observe that access to such a global transaction history also implies a Turing-complete *state machine*. Hence it can be used to realize smart contracts, which represent user-deployed programs run by the "servers" that run the blockchain protocols. Smart contracts can be used to construct custom ledgers [ERC20, ERC721], but also to facilitate more advanced functionalities such as decentralized auctions or decentralized exchanges (DEX), which forgo the need for trusted venue operators. In contrast to traditional finance, market applications in the public ledger setting offer instant settlement; any market application implemented with smart contracts instances permits the simultaneous evaluation and settlement between participants. Despite scalability challenges arising from the vast number of participants running the blockchain backbone protocol, the promise of instant settlement would allow the mitigation of counter-party risk, a real cost to transactions conducted in traditional finance today.

However, the public verifiability of a public ledger also introduces novel challenges for financial applications; account balances are public by default and leak information about submitted bids, trades or margin positions; the latter must be backed by valid balances. In decentralized finance (DeFi) [112], front-running is indeed rampant in decentralized exchanges (DEX) [109], since pending transactions leak trade intent to the adversary which can precisely order and inject transactions to execute optimal front-running strategies. Thus, proposals have been made to implement private balances on public ledgers with publicly verifiable, non-interactive zero-knowledge [99, 23]. However, a privacy-preserving ledger (even with standard smart contract support) is generally *not sufficient* for privacy-preserving financial applications such as exchanges [13].

Privacy-preserving ledgers generally complicate the realization of *smart contracts*, since these must verify and update account balances known only to its *owners* according to an agreed-upon transition logic. For decentralized exchanges implemented in the privacy-preserving ledger setting, this requires the presence of a secure multiparty computation instance, to which users can privately input their trade orders and private balances; the

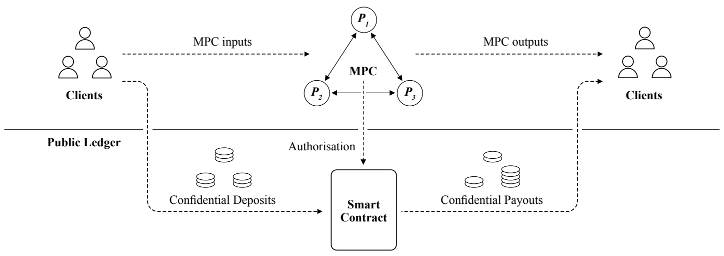


Figure 3 We sketch the architecture of privacy-preserving smart contract applications in MPC with instant settlement on a (confidential) ledger; clients provide input parameters to the MPC instance, and forward financial deposits to a smart contract in a confidential manner. The MPC privately returns computation output to clients, but also authorizes a new financial distribution which is paid out to the clients by the smart contract functionality.

MPC then computes an updated DEX state and private balances, which are then updated on the ledger (Figure 3). Enforcing consistency between the *secret*, internal MPC state and *private* account balances on the ledger requires protocol design advances illustrated in the subsequent paragraphs. We emphasize that counter to popular belief, zero-knowledge is not sufficient to realize universally expressive, privacy-preserving smart contracts, as the witness (or secret state) for decentralized privacy-preserving applications are partially held by separate, distrusting parties; instead, function evaluation over private inputs from separate parties and secret-shared data is the natural domain of secure multiparty computation.

We note there are privacy-preserving smart contract proposals which shield *private data* [104, 103] held by individual users or *private contract logic* [20], but such techniques are generally limited in their expressiveness. The work of Bowe *et al.*[20] only supports two parties, and is not widely used to realize privacy-preserving financial applications.

Sealed-bid Auctions (with Instant Settlement). The first work by Bag *et al.* [6] to realize sealed-bid auctions specifically in the setting of public ledgers focuses on using the blockchain as a *communication medium* instead of a settlement layer; as a permissionless protocol, any party can anonymously post an arbitrary message to the bulletin board, visible to all other parties. For protocols with low communication rounds, this is a practical solution; in particular, the simplicity of evaluating single-sided sealed-bid auctions permits task-specific secure multiparty protocols which only require public message broadcasts. The SEAL [6] protocol proposes the use of a *anonymous veto protocol* [69] requiring only two communication two rounds, that is then repeated once by auction bidders for *each bit* of their bid price, thereby removing the necessity an auctioneer role entirely. In particular, the veto protocol of [69] receives the private input $b_i \in \{0,1\}$ for party $i \in [n]$, for each of the m bits representing the permissible price range; the parties learn the highest bid, bit by bit. Each execution of the veto protocol will thus publicly output 1 if one of the users submits a veto; thus, by repeating the veto protocol for each bit position, all participants receive the highest bid without revealing the prices of failed bids. [6] assumes participants to behave according to the protocol (semi-honesty).

This mechanism was later adopted and hardened by FAST [49] to be secure against malicious participants not adhering to the protocol. Furthermore, [49] introduces *guaranteed settlement* on the public ledger featuring privacy-preserving deposits. Participants are thus committed to execute the payment for their bids if these are successful during the auctions. Cheating participants are penalized by having their deposits slashed and reimbursed to other parties; non-interactive zero-knowledge proofs from all parties ensure that parties only submit a veto if it is consistent with their initial bid (in commitment form on the ledger); still, despite the privacy-preserving aspects of the anonymous veto protocol, privacy leakage occurs when the highest bidder learns when he overtakes the second highest bidder. The work of Ganesh *et al.* [60] adopts a similar construction which is proven to be game-theoretically secure; it is rational to pursue the honest protocol despite any strategy chosen by other players. The work of Chin *et al.* [40] does not employ zero-knowledge proofs to shield commmitted funds for a single-sided, sealed-bid auction; deposits are sent to committed, yet undeployed contracts and are thus indistinguishable from normal Ethereum transactions, a similar technique used in Breidenbach *et al.* [21] to commit inputs to smart contracts without revealing them to the front-running adversary. This approach only guarantees k-anonymity and relies on the presence of other, unrelated transactions.

Privacy-preserving Decentralized Exchanges. We note a number of recent proposals for privacy-preserving DEX applications in recent years; intent-based privacy-preserving DEX applications mirror the functionality of over-the-counter (OTC) venues (in traditional finance) and only require a public ledger, but do not scale well and are not widely deployed. Privacy-preserving and front-running secure DEX protocols generally involve private ledger deposits and perform the order matching in an MPC instance, as is the case in distributed Dark Pool proposals previously described in Section 4.1, but offer instant settlement following each DEX round (Figure 3).

Intent-based, privacy-preserving DEX. In the works of [20] and [87], a simpler model of a decentralized exchange is implemented; a bulletin board functionality provided by a public ledger permits a “maker” to broadcast their trade intent. An interested counter-party or “taker” then directly opens an authenticated communication channel with the maker to jointly perform a privacy-preserving atomic swap on the public ledger [20]. Intent-based DEX protocols resemble over-the-counter models in traditional finance. [87] introduces a “witness key agreement” (WKA) construction which preserves the privacy of the maker’s offer; the WKA allows a taker to establish a shared secret key with a maker which has posted its order in commitment form to the ledger. The key agreement protocol succeeds if the committed, private order fulfills a relation determined by the taker. This key permits subsequent anonymous communication with the maker to finalize the transaction.

Privacy-preserving Futures DEX. An interesting example of decentralized exchanges is illustrated by Massacci *et al.* [80], which realizes a futures exchanges modelled closely after the Chicago Mercantile Exchange; here, the *future obligation* (or contract) to buy or sell a commodity is traded. The net position of a market participant is the sum of both current liquidity balances and future obligations; importantly, a party holding a future to “sell” a given commodity, must always hold sufficient liquidity to acquire the respective commodity, as it otherwise would default on its contractual obligation. Thus, a net position that falls below zero must be liquidated to protect the counter-party of any futures contract held by the liquidated party. Achieving this in a privacy-preserving manner without *revealing the net position* of a party is the goal of the work of [80].

If the net position of a participant is revealed, price manipulations could be conducted with the explicit intention of forcing the liquidation of otherwise valid positions. Thus, [80] proposes a similar scheme to [99], where the net position of each account is committed in a cryptographic accumulator. The validity of each update to the account is proven in

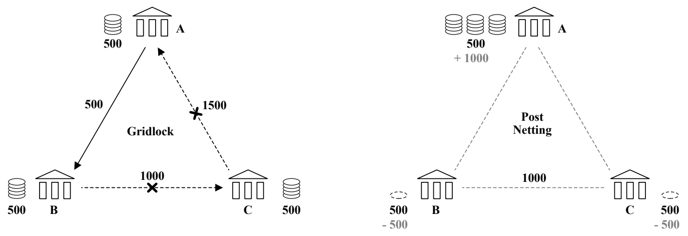


Figure 4 We adopt a netting example from [111]; processing of individual payment orders may fail due to a lack of liquidity (left), as balances must remain positive following execution of each individual payment. Netting relaxes this constraint; balances need only to be positive following execution of all payments orders (right).

zero-knowledge, whilst the trading venue is executed in a MPC instance, similarly to the Dark Pool proposals in Section 4.1. [80] requires parties participate in the protocol for each account update, even if this means the liquidation of their own account. We note that the subsequent privacy-preserving smart contract framework instantiated with MPC and a confidential ledger [9] achieves the privacy guarantees of [80] without permitting users to block application liveness.

Front-running Secure DEX. A general motivation for privacy in decentralized exchanges is the front-running of DEX applications in Decentralized Finance due to public transactions and accounts in the default ledger setting; Despite offering instant settlement of trades and transactions, pending user input authorizations generally broadcast a users trade intent before their finalization. To this end, P2DEX [11] proposes the first privacy-preserving decentralized exchange, which can operate and settle transactions across multiple ledger instances; clients submit orders to an MPC committee which computes the order matching and subsequently settles these on the respective public ledgers; since the trade inputs are private, front-running is mitigated. Follow-up work [9] generalizes this model to a setting with confidential accounts; here, all zero-knowledge proofs are moved *outside* the MPC computation, as computing such proofs *inside* the MPC remains generally unfeasible for real-world application. The work of Govindarajan *et al.* [68] realizes a privacy-preserving DEX in a similar manner; here, however, the actual order matching is computed in the clear of a smart contract over anonymized trade lists to accelerate the determination of a clearing price.

4.3 Inter-bank Netting on Public Ledgers

Inter-bank payment requests are currently submitted to the real-time gross settlement (RTGS) system managed by the central bank to update the accounts of sending and receiving financial institutions. In times of low liquidity, a bank may fail to honor *individual payment instructions*, as the liquidity requirement may exceed its balance and credit line granted by the central bank; a *gridlock* occurs, when a failed payment settlement prevents further payment instructions from being processed. Given the large payment volumes processed each day, liquidity saving mechanisms are implemented which settle payment instructions on a *netting basis* (Figure 4).

Recent work has proposed *distributing* the role of the RTGS operator with a public ledger protocol, whilst implementing efficient netting protocols with smart contracts [111, 84], thereby increasing system resiliency as the operational liability burden on the central bank operator today is very high. Whilst the aforementioned works implement inter-bank netting of queued payments, the nature of public ledgers means that payment instructions are revealed to parties participating in the underlying blockchain backbone protocol. Instead, [32] proposes payment instructions to be posted to the ledger in *commitment form* accompanied with non-interactive zero-knowledge proofs attesting their well-formedness. Here, local netting solutions are computed *by each participating bank* and verified by a coordinating smart contract, which verifies correctness of all submitted, local netting solutions without revealing amounts and the identity of institutions. Since parties must compute partial netting solutions, the protocol of [32] is not *robust* against cheating participants, who can stall or abort the netting process by posting invalid partial netting proposals. In contrast, [50] computes the netting solution inside an MPC instance, thereby achieving *fault tolerance* against dishonest participants. Despite initial implementation benchmarks provided by works above, it remains an open question in what configuration such systems can scale to real-world payment settlement volume and what netting frequency is required in practice.

5 Future applications

We will now outline which other PET use cases could be of interest in the financial sector in the foreseeable future. While many of the use cases previously described in this work may also not yet be production-ready, we want to highlight areas in this section which we think deserve more attention by researchers and practitioners. This necessarily is of speculative nature, so the reader may see this as food for thought.

Voting. Voting is a standard mechanism in deciding on future policies. While in many cases it is sufficient to make the whole voting process public, this is not always possible. For example, a voter may fear repercussions or embarrassment if his or her vote becomes public. Hence, to ensure *honest digital voting*, cryptographic voting algorithms have to be used. These ensure that election outcomes can be computed while individual votes cannot be attributed to participants. While such cryptographic voting can be realized using MPC or FHE, a dedicated line of work started by Chaum [38] presents highly efficient dedicated voting protocols. Cryptographic voting mechanisms find interesting applications in the DeFi space, e.g. for privacy-preserving Decentralized Anonymous Organizations (DAOs). In particular, a treasury system for DAOs based on electronic voting has been proposed in [116] and a board room voting scheme based on smart contracts (and this amenable to the DAO scenario) has been proposed in [82]. We believe that these techniques may also be useful for coordination among classical banking institutions and other financial operations (*e.g.* shareholder meetings).

Fraud detection. Both insurance and gambling are known as industries where companies in the sector exchange information on their customers in order to detect fraud or exploitative customers[11]. This information sharing may be problematic for privacy reasons, and it also leaks information about suspected but ultimately honest customers if done in plain. PETs such as PSI might be an interesting tool to construct a trusted intermediary. This intermediary can obtain information from participating companies and alerts them if e.g. more than 3 of them share the same customer. Here, PSI can ensure that only those customers are revealed that appear often enough.

[11] For example the infamous “Griffin Book”.

Better and fairer pattern recognition. In section 3 we have outlined how AML does benefit from recognition of suspicious patterns. Such patterns, if one wants to keep up in the digital age, must be learned from a large dataset and must be updated frequently. Moreover, many companies in an industry have an interest in pooling their data with other institutions for the purpose of learning these patterns. At the same time, they may not want to share raw customer or transaction data. Another, related area is assessing the credit risk of potential customers. Here, the risk becomes more accurate the more participants can contribute information or models. At the same time, input providers have an interest to keep their data private (for data protection or to protect intellectual property).

Both applications fall into the area of privacy-preserving Machine Learning [78, 54, 83] or confidential benchmarking [47] which are subfields of MPC. While these areas have received much attention recently[12], optimized applications to finance seem to be lacking.

Another important aspect is that (automatically generated) models should not be biased against certain groups. While fair machine learning itself is a rapidly developing field, its application to finance [51] may deserve more attention.

Privacy preserving mitigation of systemic risk. Audits of financial institutions guarantee that their balances plus credit cover outstanding obligations. This reduces counter-party risk and means that the overall system can rely less on biasable methods such as ratings and reputation. At the same time, an audited company may not want to open its books fully to the public, or it might not be guaranteed that these books are correct. [80] have shown how audits can be realized using ZK proofs, although limited to the futures market. We believe that this concept may be generalized to the wider financial system to permit privacy-preserving audits.

References

1 Aysajan Abidin, Abdelrahaman Aly, Sara Cleemput, and Mustafa A Mustafa. An mpc-based privacy-preserving protocol for a local electricity trading market. In *Cryptology and Network Security: 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings 15*, pages 615–625. Springer, 2016.

2 Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. PASTA: PASsword-based threshold authentication. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 2042–2059. ACM Press, October 2018. doi:10.1145/3243734.3243839.

3 Ghada Almashaqbeh and Ravital Solomon. Sok: Privacy-preserving computing in the blockchain era. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 124–139, 2022. doi:10.1109/EuroSP53844.2022.00016.

4 Nasser Alsalami and Bingsheng Zhang. Sok: A systematic study of anonymity in cryptocurrencies. In *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–9, 2019. doi:10.1109/DSC47296.2019.8937681.

5 Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. Privacy-preserving auditable token payments in a permissioned blockchain system. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 255–267. ACM, 2020. doi:10.1145/3419614.3423259.

6 Samiran Bag, Feng Hao, Siamak F Shahandashti, and Indranil Ghosh Ray. SEAL: Sealed-bid auction without auctioneers. *IEEE Transactions on Information Forensics and Security*, 15:2042–2052, 2019.

[12] Privacy-preserving Machine Learning opens up interesting use cases, but it does not come without its own problems. See [52] for a good overview.

7 Tucker Balch, Benjamin E Diamond, and Antigoni Polychroniadou. SecretMatch: inventory matching from fully homomorphic encryption. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–7, 2020.

8 Amira Barki and Aline Gouget. Achieving privacy and accountability in traceable digital currency. Cryptology ePrint Archive, Report 2020/1565, 2020. URL: `https://eprint.iacr.org/2020/1565`.

9 Carsten Baum, James Hsin-yu Chiang, Bernardo David, and Tore Kasper Frederiksen. Eagle: Efficient Privacy Preserving Smart Contracts. *Cryptology ePrint Archive (To appear in Financial Cryptography and Data Security 2023)*, 2022. URL: `https://eprint.iacr.org/2022/1435`.

10 Carsten Baum, James Hsin-yu Chiang, Bernardo David, and Tore Kasper Frederiksen. SoK: Privacy-Enhancing Technologies in Finance, 2023. Full version. URL: `https://eprint.iacr.org/2023/122`.

11 Carsten Baum, Bernardo David, and Tore Kasper Frederiksen. P2DEX: privacy-preserving decentralized cryptocurrency exchange. In *International Conference on Applied Cryptography and Network Security*, pages 163–194. Springer, 2021.

12 Carsten Baum, Tore Kasper Frederiksen, Julia Hesse, Anja Lehmann, and Avishay Yanai. PESTO: proactively secure distributed single sign-on, or how to trust a hacked server. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 587–606. IEEE, 2020. `doi:10.1109/EuroSP48549.2020.00044`.

13 Carsten Baum, James Hsin yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. Sok: Mitigation of front-running in decentralized finance. Cryptology ePrint Archive, Paper 2021/1628, 2021. To appear on the Proceedings of the The 2nd Workshop on Decentralized Finance (DeFi) in Association with Financial Cryptography 2022. URL: `https://eprint.iacr.org/2021/1628`.

14 Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988. `doi:10.1145/62212.62213`.

15 Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014. `doi:10.1109/SP.2014.36`.

16 Abhishek Bhowmick, Dan Boneh, Steve Myers, Kunal Talwar, and Karl Tarbe. The apple PSI system, 2021. Accessed on 02/08/2023. URL: `https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf`.

17 Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012. `doi:10.1145/2090236.2090263`.

18 Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009*, volume 5628 of *LNCS*, pages 325–343. Springer, Heidelberg, February 2009.

19 Peter Bogetoft, Ivan Damgård, Thomas Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. A practical implementation of secure auctions based on multiparty integer computation. In Giovanni Di Crescenzo and Avi Rubin, editors, *FC 2006*, volume 4107 of *LNCS*, pages 142–147. Springer, Heidelberg, February / March 2006.

20 Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964. IEEE, 2020.

21 Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. Enter the hydra: Towards principled bug bounties and {Exploit-Resistant} smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1335–1352, 2018.

22 Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 423–443. Springer, Heidelberg, February 2020. doi:10.1007/978-3-030-51280-4_23.

23 Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 2020.

24 Joseph Burleson, Michele Korver, and Dan Boneh. Privacy-protecting regulatory solutions using zero-knowledge proofs: Full paper, 2020. URL: https://a16zcrypto.com/privacy-protecting-regulatory-solutions-using-zero-knowledge-proofs-full-paper/.

25 Christian Cachin. Efficient private bidding and auctions with an oblivious third party. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 120–127, 1999.

26 Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 683–699. ACM Press, October / November 2017. doi:10.1145/3133956.3134025.

27 Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 345–356. ACM Press, October 2008. doi:10.1145/1455770.1455814.

28 Jan Camenisch, Anja Lehmann, Anna Lysyanskaya, and Gregory Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 256–275. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44381-1_15.

29 Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001. doi:10.1007/3-540-44987-6_7.

30 Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004. doi:10.1007/978-3-540-28628-8_4.

31 Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 21–30. ACM Press, November 2002. doi:10.1145/586110.586114.

32 Shengjiao Cao, Yuan Yuan, Angelo De Caro, Karthik Nandakumar, Kaoutar Elkhiyaoui, and Yanyan Hu. Decentralized privacy-preserving netting protocol on blockchain for payment systems. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 137–155, Cham, 2020. Springer International Publishing.

33 John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. MPC joins the dark side. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 148–159, 2019.

34 John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. Multi-party computation mechanism for anonymous equity block trading: A secure implementation of turquoise plato uncross. *Intelligent Systems in Accounting, Finance and Management*, 28(4):239–267, 2021.

35 Benjamin Case, Richa Jain, Alex Koshelev, Andy Leiserson, Daniel Masny, Thurston Sandberg, Ben Savage, Erik Taubeneck, Martin Thomson, and Taiki Yamaguchi. Interoperable private attribution: A distributed attribution and aggregation protocol. Cryptology ePrint Archive, Paper 2023/437, 2023. URL: https://eprint.iacr.org/2023/437.

36 David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract) (informal contribution). In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, page 462. Springer, Heidelberg, August 1988. doi:10.1007/3-540-48184-2_43.

37 David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988. doi:10.1145/62212.62214.

38 David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

39 Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1243–1255. ACM Press, October / November 2017. doi:10.1145/3133956.3134061.

40 Kota Chin, Keita Emura, Kazumasa Omote, and Shingo Sato. A Sealed-bid Auction with Fund Binding: Preventing Maximum Bidding Price Leakage. In *2022 IEEE International Conference on Blockchain (Blockchain)*, pages 398–405. IEEE, 2022.

41 Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th FOCS*, pages 383–395. IEEE Computer Society Press, October 1985. doi:10.1109/SFCS.1985.64.

42 Coinbase. Coinbase to acquire leading cryptographic security company, Unbound Security, November 2021. URL: https://www.coinbase.com/blog/coinbase-to-acquire-leading-cryptographic-security-company-unbound-security.

43 d1onys1us. Zero-knowledge rollups, 2023. Accessed on 02/08/2023. URL: https://ethereum.org/en/developers/docs/scaling/zk-rollups/.

44 Mariana Botelho da Gama, John Cartlidge, Antigoni Polychroniadou, Nigel P Smart, and Younes Talibi Alaoui. Kicking-the-bucket: Fast privacy-preserving trading using buckets. In *International Conference on Financial Cryptography and Data Security*, pages 20–37. Springer, 2022.

45 Mariana Botelho da Gama, John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. All for one and one for all: Fully decentralised privacy-preserving dark pool trading using multi-party computation. *Cryptology ePrint Archive*, 2022. URL: https://eprint.iacr.org/2022/923.

46 Mariana Botelho da Gama, John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. Privacy-preserving dark pools. *Cryptology ePrint Archive*, 2022. URL: https://eprint.iacr.org/2022/923.

47 Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential benchmarking based on multiparty computation. In Jens Grossklags and Bart Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 169–187. Springer, Heidelberg, February 2016.

48 Ivan Damgård, Chaya Ganesh, Hamidreza Khoshakhlagh, Claudio Orlandi, and Luisa Siniscalchi. Balancing privacy and accountability in blockchain identity management. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 552–576. Springer, Heidelberg, May 2021. doi:10.1007/978-3-030-75539-3_23.

49 Bernardo David, Lorenzo Gentile, and Mohsen Pourpouneh. FAST: fair auctions via secret transactions. In *International Conference on Applied Cryptography and Network Security*, pages 727–747. Springer, 2022.

50 Angelo De Caro, Andrew Miller, and Amit Agarwal. Privacy-Preserving Decentralized Multi-Party Netting, September 29 2022. US Patent App. 17/216,644, https://patents.google.com/patent/US20220309492A1/en.

51 Leo de Castro, Jiahao Chen, and Antigoni Polychroniadou. Cryptocredit: securely training fair models. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020.

52 Emiliano De Cristofaro. A critical overview of privacy in machine learning. *IEEE Security & Privacy*, 19(4):19–27, 2021. doi:10.1109/MSEC.2021.3076443.

53 Apple Differential Privacy Team. Learning with privacy at scale. Accessed on 02/08/2023. URL: `https://docs-assets.developer.apple.com/ml-research/papers/learning-with-privacy-at-scale.pdf`.

54 Wenliang Du, Mikhail J Atallah, et al. Privacy-preserving cooperative scientific computations. In *csfw*, volume 1, page 273, 2001.

55 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 265–284. Springer, Heidelberg, March 2006. `doi:10.1007/11681878_14`.

56 Matthew K Franklin and Michael K Reiter. The design and implementation of a secure auction service. *IEEE Transactions on Software Engineering*, 22(5):302–312, 1996.

57 Tore Kasper Frederiksen. A holistic approach to enhanced security and privacy in digital health passports. In Delphine Reinhardt and Tilo Müller, editors, *ARES 2021: The 16th International Conference on Availability, Reliability and Security, Vienna, Austria, August 17-20, 2021*, pages 133:1–133:10. ACM, 2021. `doi:10.1145/3465481.3469212`.

58 Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004. `doi:10.1007/978-3-540-24676-3_1`.

59 Mariana Gama, Fairouz Zobiri, and Svetla Nikova. Multi-party computation auction mechanisms for a p2p electricity market with geographical prioritization, 2022. URL: `https://www.esat.kuleuven.be/cosic/publications/article-3526.pdf`.

60 Chaya Ganesh, Bhavana Kanukurthi, and Girisha Shankar. Secure Auctions in the Presence of Rational Adversaries. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1173–1186, 2022.

61 Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 281–310. Springer, 2015.

62 Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. In *NDSS 2014*. The Internet Society, February 2014.

63 Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009. `doi:10.1145/1536414.1536440`.

64 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

65 Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th FOCS*, pages 174–187. IEEE Computer Society Press, October 1986. `doi:10.1109/SFCS.1986.47`.

66 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. `doi:10.1145/28395.28420`.

67 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985. `doi:10.1145/22145.22178`.

68 Kavya Govindarajan, Dhinakaran Vinayagamurthy, Praveen Jayachandran, and Chester Rebeiro. Privacy-preserving decentralized exchange marketplaces. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2022.

69 Feng Hao and Piotr Zieliński. A 2-round anonymous veto protocol. In *International Workshop on Security Protocols*, pages 202–211. Springer, 2009.

70 Michael Harkavy, J Doug Tygar, and Hiroaki Kikuchi. Electronic auctions with private bids. In *USENIX Workshop on Electronic Commerce*, 1998.

71 Space systems — Definition of the Technology Readiness Levels (TRLs) and their criteria of assessment. Standard, International Organization for Standardization, Geneva, CH, November 2013.

72 Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 39–58. Springer, Heidelberg, July 2017. doi:10.1007/978-3-319-61204-1_3.

73 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017.

74 Markulf Kohlweiss, Anna Lysyanskaya, and An Nguyen. Privacy-preserving blueprints. *IACR Cryptol. ePrint Arch.*, page 1536, 2022. URL: https://eprint.iacr.org/2022/1536.

75 Team KZen. Bitcoin wallet powered by two-party ECDSA extended abstract. URL: https://github.com/ZenGo-X/gotham-city/blob/master/white-paper/white-paper.pdff.

76 Protocol Labs. Filecoin: A decentralized storage network, 2017. URL: https://filecoin.io/filecoin.pdf.

77 Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1837–1854, 2018.

78 Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 36–54. Springer, Heidelberg, August 2000. doi:10.1007/3-540-44598-6_3.

79 Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. CanDID: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *2021 IEEE Symposium on Security and Privacy*, pages 1348–1366. IEEE Computer Society Press, May 2021. doi:10.1109/SP40001.2021.00038.

80 Fabio Massacci, Chan Nam Ngo, Jing Nie, Daniele Venturi, and Julian Williams. FuturesMEX: secure, distributed futures market exchange. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 335–353. IEEE, 2018.

81 Nick Maxwell. Case studies of the use of privacy preserving analysis to tackle financial crime, January 2021. URL: https://www.future-fis.com/uploads/3/7/9/4/3794525/ffis_innovation_and_discussion_paper_-_case_studies_of_the_use_of_privacy_preserving_analysis_-_v.1.3.pdf.

82 Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. In Aggelos Kiayias, editor, *FC 2017*, volume 10322 of *LNCS*, pages 357–375. Springer, Heidelberg, April 2017.

83 Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society Press, May 2017. doi:10.1109/SP.2017.12.

84 Ken Naganuma, Masayuki Yoshino, Hisayoshi Sato, Nishio Yamada, Takayuki Suzuki, and Noboru Kunihiro. Decentralized netting protocol over consortium blockchain. In *2018 International Symposium on Information Theory and Its Applications (ISITA)*, pages 174–177. IEEE, 2018.

85 Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, 1999.

86 Neha Narula, Willy Vasquez, and Madars Virza. zkLedger: Privacy-Preserving Auditing for Distributed Ledgers. In Sujata Banerjee and Srinivasan Seshan, editors, *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, pages 65–80. USENIX Association, 2018. URL: https://www.usenix.org/conference/nsdi18/presentation/narula.

87 Chan Nam Ngo, Fabio Massacci, Florian Kerschbaum, and Julian Williams. Practical witness-key-agreement for blockchain-based dark pools financial trading. In *International Conference on Financial Cryptography and Data Security*, pages 579–598. Springer, 2021.

88 Christian Paquin. U-Prove Technology Overview V1.1. Tech report, Microsoft Corporation, April 2013. URL: `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/U-Prove20Technology20Overview20V1.120Revision202.pdf`.

89 David C Parkes, Michael O Rabin, Stuart M Shieber, and Christopher Thorpe. Practical secrecy-preserving, verifiably correct and trustworthy auctions. *Electronic Commerce Research and Applications*, 7(3):294–312, 2008.

90 THE EUROPEAN PARLIAMENT and THE COUNCIL OF THE EUROPEAN UNION. Directive (EU) 2015/2366 of the european parliament and of the council, November 2015. URL: `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32015L2366&from=EN`.

91 Juha Partala, Tri Hong Nguyen, and Susanna Pirttikangas. Non-interactive zero-knowledge for blockchain: A survey. *IEEE Access*, 8:227945–227961, 2020. `doi:10.1109/ACCESS.2020.3046025`.

92 Alexey Pertsev, Roman Semenov, and Roman Storm. Tornado Cash Privacy Solution, version 1.4, December 2019. URL: `https://web.archive.org/web/20211026053443/https://tornado.cash/audits/TornadoCash_whitepaper_v1.4.pdf`.

93 Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989. `doi:10.1145/73007.73014`.

94 Peter Reuter and Edwin M. Truman. *Chasing Dirty Money: The Fight Against Money Laundering.* Peterson Institute for International Economics, 2004.

95 Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

96 Umut Sahin. How to get started with concrete - zama's fully homomorphic encryption compiler, 2023. Accessed on 02/08/2023. URL: `https://www.zama.ai/post/how-to-started-with-concrete-zama-fully-homomorphic-encryption-compiler`.

97 Tomas Sander and Amnon Ta-Shma. Flow control: A new approach for anonymity control in electronic cash systems. In Matthew Franklin, editor, *FC'99*, volume 1648 of *LNCS*, pages 46–61. Springer, Heidelberg, February 1999.

98 Olivier Sanders. Efficient redactable signature and application to anonymous credentials. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 628–656. Springer, Heidelberg, May 2020. `doi:10.1007/978-3-030-45388-6_22`.

99 Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014.

100 Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

101 Manu Sporny, Dave Longley, and David Chadwick. Verifiable credentials data mode, 2022. URL: `https://www.w3.org/TR/vc-data-model/`.

102 Manu Sporny, Dave Longley, Markus Sabadello, Drummond Reed, Orie Steele, and Christopher Allen. Decentralized identifiers (DIDs), 2022. URL: `https://www.w3.org/TR/did-core`.

103 Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin Vechev. ZeeStar: Private Smart Contracts by Homomorphic Encryption and Zero-knowledge Proofs. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1543–1543. IEEE Computer Society, 2022.

104 Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1759–1776, 2019.

105 Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. Mind your weight(s): A large-scale study on insufficient machine learning model protection in mobile apps. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 1955–1972. USENIX Association, August 2021.

106 Shahroz Tariq, Sowon Jeon, and Simon S. Woo. Am I a real or fake celebrity? evaluating face recognition and verification apis under deepfake impersonation attack. In Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 512–523. ACM, 2022. `doi:10.1145/3485447.3512212`.

107 Christopher Thorpe and David C Parkes. Cryptographic securities exchanges. In *International Conference on Financial Cryptography and Data Security*, pages 163–178. Springer, 2007.

108 Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. UTT: Decentralized ecash with accountable privacy. Cryptology ePrint Archive, Report 2022/452, 2022. URL: `https://eprint.iacr.org/2022/452`.

109 Christof Ferreira Torres, Ramiro Camino, and Radu State. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1343–1359. USENIX Association, 2021. URL: `https://www.usenix.org/conference/usenixsecurity21/presentation/torres`.

110 Marie Beth van Egmond, Thomas Rooijakkers, and Alex Sangers. Privacy-Preserving Collaborative Money Laundering Detection. *ERCIM News*, 2021(126), 2021. URL: `https://ercim-news.ercim.eu/en126/special/privacy-preserving-collaborative-money-laundering-detection`.

111 Xin Wang, Xiaomin Xu, Lance Feagan, Sheng Huang, Limei Jiao, and Wei Zhao. Inter-bank payment system on enterprise blockchain platform. In *2018 IEEE 11th international conference on cloud computing (CLOUD)*, pages 614–621. IEEE, 2018.

112 Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. Sok: Decentralized finance (defi). *arXiv preprint arXiv:2101.08778*, 2021. URL: `https://arxiv.org/abs/2101.08778`.

113 Karl Wüst, Kari Kostiainen, Vedran Capkun, and Srdjan Capkun. PRCash: Fast, private and regulated transactions for digital currencies. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 158–178. Springer, Heidelberg, February 2019. `doi:10.1007/978-3-030-32101-7_11`.

114 Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. Veri-zexe: Decentralized private computation with universal setup. *Cryptology ePrint Archive*, 2022.

115 Arman Zand, James Orwell, and Eckhard Pfluegel. A Secure Framework for Anti-Money-Laundering using Machine Learning and Secret Sharing. In *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–7, 2020. `doi:10.1109/CyberSecurity49315.2020.9138889`.

116 Bingsheng Zhang, Roman Oliynykov, and Hamed Balogun. A treasury system for cryptocurrencies: Enabling better collaborative intelligence. In *NDSS 2019*. The Internet Society, February 2019.

117 Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 270–282. ACM Press, October 2016. `doi:10.1145/2976749.2978326`.

118 Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: Liberating web data using decentralized oracles for TLS. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1919–1938. ACM Press, November 2020. `doi:10.1145/3372297.3417239`.

119 Fairouz Zobiri, Mariana Gama, Svetla Nikova, and Geert Deconinck. A Privacy-Preserving Three-Step Demand Response Market Using Multi-Party Computation. In *13th Int. Conf. Innov. Smart Grid Technol.(ISGT North Am. 2022), Washingt. DC (to Appear)*, 2022. URL: `https://www.esat.kuleuven.be/cosic/publications/article-3451.pdf`.

Decentralization Cheapens Corruptive Majority Attacks

Stephen H. Newman
Princeton University, NJ, USA

Abstract

Corruptive majority attacks, in which mining power is distributed among miners and an attacker attempts to bribe a majority of miners into participation in a majority attack, pose a threat to blockchains. Budish bounded the cost of bribing miners to participate in an attack by their expected loss as a result of attack success. We show that this bound is loose. In particular, an attack may be structured so that under equilibrium play by most miners, a miner's choice to participate only slightly affects the attack success chance. Combined with the fact that most of the cost of attack success is externalized by any given small miner, this implies that if most mining power is controlled by small miners, bribing miners to participate in such an attack is much cheaper than the Budish bound. We provide a scheme for a cheap corruptive majority attack and discuss practical concerns and consequences.

2012 ACM Subject Classification Theory of computation → Algorithmic mechanism design; Applied computing → Digital cash

Keywords and phrases Blockchain, Majority Attack, Corruptive Majority Attack

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.13

Acknowledgements Thanks to Matt Weinberg for substantial discussion, feedback, and advice.

1 Introduction

Blockchain- and consensus-based ledger protocols are generally susceptible to *majority attacks*, in which an attacker gains control of a majority of mining power and uses it to create a new canonical transaction history which diverges substantially from the original heaviest chain [11]. This attack may be highly profitable: on currency-only blockchains, this enables doublespend attacks and may cause chaos and/or devaluation, all of which may be used for economic gain. On blockchains that implement higher-level protocols and applications, such as Ethereum, attackers may also retroactively alter the state of smart contracts or other time-varying constructs, with similar consequences.

Historically, concerns about majority attacks have been dismissed as irrelevant to the current state of major cryptocurrencies. Budish argues that this is not necessarily the case in the long run (or even currently): the cost of an attack on the blockchain is proportional to the mining payout rate, so large transaction flow relative to this cost makes majority attacks profitable [3]. We show that under simple assumptions about the distribution of miner powers, the situation is far *worse* than Budish's upper bound suggests. Budish bounds the necessary payout to miners as the cost to them of attack success. While the cost of attack success is indeed well-estimated by Budish, individual miners' actions are typically not substantially causally correlated with attack success, and so it suffices to pay miners their cost of attack success *times the marginal increase in attack success probability which resulted from their actions.* As a result, in the by-design scenario where miners are small and therefore no individual or small group can exercise substantial control over a blockchain (though this is not always the stable state of affairs [1]), corruptive majority attacks are both cheap and hard to prevent, implying that cryptocurrencies are less game-theoretically stable than previously believed.

After a brief exposition of our model, we develop a simple framework for miner incentive analysis that illuminates the action/result-correlation aspect of their incentive problem. In particular, we show that for small miners, costs of participation in an attack are mostly externalized and sometimes very small. We then develop a more rigorous model of corruptive attacks, in which miners can change their behavior from timestep to timestep depending on state of the attack, and show that appropriately structured attacks succeed and are cheap with high probability. We give a practical example of such an attack on Bitcoin, including calculation of expected cost and discussion of profitability. We discuss novel and previously proposed economic-incentive-based prevention and mitigation strategies. We also give a brief overview of some of the noneconomic incentives that affect the feasibility/likelihood of such an attack.

Related Work

The consideration of majority attacks against digital currencies dates back to at least Nakamoto's work [11]. Bonneau noted the danger of bribery-based attacks and discussed attack methodologies and potential countermeasures in [2]. Since then, a variety of game-theoretic attack techniques (e.g. [8]) and practical methodologies (e.g. [9]) have been proposed. Judmeyer et al. provide a broad overview of bribery attack modeling in [7], including majority attacks. Several majority attacks have also been conducted against a variety of cryptocurrencies, with varying results, as summarized (as of 2019) by [12].

The central – and often ignored – obstacle in conducting a majority attack is the long-term cost to miners resulting from currency devaluation stemming from the attack, as first noted in [11]. Budish analyzed this concept in detail, providing a formal model of the cost to a majority miner and approximating it for Bitcoin in [3]. Moroz et al. responded with a model of an attack-counterattack game with a liquid mining power marketplace, arguing that in the Budish setting, the threat of counterattack served to deter majority attacks designed to allow doublespending [10].

There is also growing interest in the effects of incentive manipulation on a wider class of social-choice mechanisms without money. For instance, bribery [6] and coalition effects [14] have been studied in the context of voting, and there is continuing investigation into similar effects in matching markets and other contexts.

2 Modeling Mining and the Cost of Corruption

2.1 Mining Model

We assume a model of mining and miner incentives similar to that of Budish [3]. We assume a fixed set M of miners, with a function $\phi : M \to \mathbb{R}_{\geq 0}$ mapping miners to their powers (hashrates in the case of hash-based PoW coins, for instance), and denote the total power $\Phi \overset{\text{def}}{=} \sum_{m \in M} \phi(m)$. We assume that each block mined is mined by a random miner, chosen at time of mining, with independent probability $\frac{\phi(m)}{\Phi}$ of selecting miner m to mine any given block. A miner who mines a block receives reward R for doing so. We assume that miners may choose to dedicate mining power to an attacker whose goal is to execute a majority attack. For simplicity (and optimistically for stability against such attacks), we assume that miners who attack on a mining turn receive no block rewards that turn.

2.2 Miner Valuation Model

We assume that miners extract value exclusively from present and future mining activity. In particular, we will consider two primary costs to miners of selling mining power: the direct expected lost revenue from lost time mining, and the expected lost future revenue from the increased chance of a majority attack as a result of providing hashpower. We will assume (pessimistically for the attacker) that a successful majority attack will cost a miner exactly their time-discounted expected future revenue[1]. This will be indicated by vR for some value v to be bounded later.

As in Budish's work, the primary cost of attacks will lie in compensating miners for the change in the likelihood of attack success due to their participation – the cost of corruption. We wish to bound this.

2.3 Budish's Cost of Corruption

Budish argued (and we agree) that the cost to miners caused by an attack is bounded by a sum of two costs. The more obvious component is their expected *immediate loss*: their lost mining revenue as a result of devoting mining time to the attack. This may be trivially upper-bounded as

$$T\frac{\phi(m)}{\Phi}R$$

for an attack stretching over T timesteps, where R is the block reward, and will be negligible compared to the second component of the loss in most cases.

The more subtle loss which a miner incurs is their expected *devaluation loss*: their time-discounted future loss of revenue as a result of currency collapse. Following Budish's analysis, we may bound this by

$$v\frac{\phi(m)}{\Phi}R$$

for some constant multiplier v.[2] For instance, assuming constant distribution of mining power for a cryptocurrency which pays k fixed block rewards R per year, $v = \frac{1}{1-\sqrt[k]{0.95}} \approxeq 20k$ would reflect the sum of expected income for all time with 5%/year discounting applied. In most cases, however, v is substantially lower. For instance, assuming that R halves every few years (or that the total mining power of other miners is large and doubles every few years) we may bound v as the number of blocks in just a few years – and assumptions about ongoing costs of mining and revenue from selling mining equipment in a crash further reduce this estimate.

Combining the above two losses, we may weakly incentivize a miner $m \in M$ to participate in an attack of duration T by paying them $(T+v)\frac{\phi(m)}{\Phi}R$. Summing across miners, we see that it costs

$$(T+v)R$$

to pay all miners to participate.

[1] This corresponds to the assumption that a successful attack will prevent any future mining, while a failed attack does not change the value of future mining, causing the largest possible losses to miners.

[2] We assume that a coin will have no change in value under attack failure and will undergo total and permanent devaluation under attack success. This represents an optimistic assumption from the point of view of stability, as it drives the expected cost to miners of an attack (and therefore the cost of said attack) as high as possible.

As Budish points out, this may already be dangerous. TR is low compared to potential profit, and v may be as well – for blockchains where Φ is expected to rise rapidly (PoW coins, for instance), gross mining income for given hardware is expected to drop proportionately, implying that almost all income is obtained in the next few years of mining. For instance, assuming $v \approxeq 2 \cdot 10^5$, the number of Bitcoin blocks in a little under four years, we get an attack price above $1.2 \cdot 10^6$ BTC, slightly over 5% of BTC in existence. Even in cases where mining power does not substantially increase (some PoS protocols, for instance) and currency valuation is expected to remain approximately constant, v may be bounded by the number of blocks in twenty years, as $v\frac{\phi(m)}{\Phi}R$ is then enough to purchase a stock market portfolio which, if it yields 5%/year over inflation, will be more profitable than mining at relative power $\frac{\phi(m)}{\Phi}$ assuming constant currency valuation.

2.4 The Participation-Success Matrix

We first consider a binary model for action during an attack: during an attack, a player p may either *participate* or *refuse* to participate, and the attack will succeed or fail with probabilities determined by the participation/refusal of the various miners. Player p's reward will depend on both participation/refusal and success/failure.

We note something simple but interesting: when a player's reward depends mostly on success or failure, and success/failure is only slightly affected by participation/refusal, the marginal cost of participation/refusal is much lower than the difference in reward between the success and failure cases. Consider, for instance, a small miner who participates in a majority attack which is expected to permanently crash Bitcoin if it succeeds. Assume that their expected time-discounted future valuation was U, and that their expected mining reward over the course of the attack period is $u << U$. Then, excluding order-u increases in payout due to difficulty reduction, their mining payoff matrix is

	Attack succeeds	Attack fails
Participate	0	U
Refuse	u	$U + u$

For instance, considering the cases where the miner is sure that the attack will (succeed/-fail) regardless of their action, their expected loss as a result of participating in the attack is bounded by $(0/u)$. Relaxing this slightly, if they are sure that their participation affects attack success chance by at most ϵ, their expected losses are at most $u + \epsilon U$.

2.5 Bounding Expected Devaluation Loss

The difference between our bound on cost of corruption and Budish's is simple: while Budish proposed paying all miners their expected losses from an attack, we propose paying them only the *marginal increase* in their expected losses as a result of their participation in the attack. We consider an explicitly multiparty attack: the attacker attempts to corrupt many smaller miners into attack participation, and therefore must pay them only the damage they expect to cost themselves by joining. Under this view, Budish's cost of corruption is not tight – it pays miners enough to convince them to join even in a case where act/no-act and success/failure are perfectly correlated. This is generally far from the truth: the smaller a miner is, the less impact their choice has on their perceived likelihood of attack success (and therefore their expected loss).

Let $f(\frac{\phi(m)}{\Phi}) : [0,1] \to [0,1]$ be a bound on the probability, as estimated by a miner m, that miner m's participation in the attack will cause it to succeed (i.e. that it will not succeed if they do not participate, and will succeed if they do). Then we may tighten the bound on their devaluation loss to $f(\frac{\phi(m)}{\Phi})\frac{\phi(m)}{\Phi}Rv$ and the bound on their total loss to

$$\left(T + f\left(\frac{\phi(m)}{\Phi}\right)v\right)\frac{\phi(m)}{\Phi}R$$

3 A Thresholding Corruptive Attack

We now analyze the attack implied by the above payout rule. Assume for the moment that we can verify full participation in an attack. Pick a start time for the attack, and pay any miner m who participates in the attack

$$(T + f_{\max}v)\frac{\phi(m)}{\Phi}R$$

for $f_{\max}$ chosen such that $\sum_{m:f\left(\frac{\phi(m)}{\Phi}\right)\leq f_{\max}} \phi(m)$ is substantially greater than $\frac{\Phi}{2}$. If each miner acts rationally, any miner m with $f(\frac{\phi(m)}{\Phi}) \leq f_{\max}$ will participate, and our attack will succeed. Summing across all miners, total cost of the attack is bounded by

$$(T + f_{\max}v)\,R$$

which for small values of $f_{\max}$ is much smaller than the attack cost under Budish's analysis.

3.1 Estimating v

v depends on changing economic conditions, the specifics of the protocol under attack, and other real-world factors [3]. In Bitcoin and other PoW coins, for instance, v equal to the number of blocks in a few years (i.e. $\frac{\phi(m)}{\Phi}vR$ equal to undiscounted gross earnings over the next few years) may be a reasonable estimate, given reward halving, electricity costs, continual improvements in mining hardware, and regulatory concerns. Notably, payouts from attack participation provide substantial security as compared to mining, for the same real-world reasons as cited above. For more in-depth discussion of v, see [3].

3.2 Estimating f

We have two means of bounding f. First, under the assumption that $\sum_{m\in M:\frac{\phi(m)}{\Phi}\leq f_{\max}} \frac{\phi(m)}{\Phi}$ is substantially greater than $\frac{1}{2}$, $f\left(\frac{\phi(m)}{\Phi}\right)$ is likely bounded on the close order of $\frac{\phi(m)}{\Phi}$, as the expected distribution of participants should not concentrate except possibly at nearly all players participating or nearly no players participating (in which case $f(m)$ is very low). As such, $f_{\max}$ is only required to be big enough to make the above fraction substantially greater than $\frac{1}{2}$, and will therefore be small for a well-decentralized blockchain. The cheaper attack price that results from smaller miners under this bound corresponds to the fact that these smaller miners externalize more of the damage caused by participating – a miner's expected loss from participation is quadratic in their power, but their expected damage is linear.

Second, and more concerningly, if we can convince all miners that the attack is destined to succeed, f becomes 0, as no individual miner's choice will affect the attack. Convincing miners that the attack will almost surely succeed likewise forces f very low.

Both of these dynamics will prove relevant in the next model.

4 Refining the Model: A Block-By-Block Attack

The previous model relied on the assumption that miner behavior over the course of an attack will not change, and assumed a somewhat arbitrary bound on f. We now consider a multiparty refinement of the classical model for majority attacks. An attack is modeled as a T-step game with chance. The state of the game at the conclusion of timestep $t \in \{1, \dots, T\}$ is given by an integer l_t indicating the length discrepancy between the attacking and defending chains (positive if the defending chain is longer), with l_0 the distance from the attack fork point to the tip of the heaviest chain at the start of the attack.

The attacker wins the game at the first timestep t where $l_t = 0$; if no such timestep has occurred by $t = T$, the attacker loses. As in [7] (in the case of zero native attacker power), miners are partitioned into two groups. We assume that a substantial set of the miners are honest (will always defend), either because we have not provided them sufficient economic incentive to attack, as will be the case for very large miners, or because of non-economic factors. We will model the remaining non-committed miners as rational agents that choose to mine for either the attacker or the defender based on expected time-discounted future rewards. At each step t, all miners decide to either attack or defend, and a random miner is chosen with probability proportional to their mining power. If that miner is defending, they mine a block on the canonical chain, setting $l_t = l_{t-1} + 1$. If attacking, they mine a block on the attacking chain, setting $l_t = l_{t-1} - 1$, and they receive a payout $c_{t-1,l}$ from the attacker for their choice to attack. Equivalently, we may model miners as making the choice to attack/defend after they are selected as the current round's block miner. At the end of the game, if the attacker lost, each miner m receives payout $v\frac{\phi(m)}{\Phi}R$, their expected future mining returns.

Our payout rule will not incentivize participation by miners of size $> \gamma\Phi$, so we assume for simplicity that all such miners are part of the honest pool – though if they do, it will aid the attack. We assume that the honest miners have combined mining power $\leq g_{\mathrm{Def}}\Phi$. We will again exploit the fact that the actions of small miners (of power $\leq \gamma\Phi$, for our purposes) are only slightly correlated with attack success chance to construct a cheap attack.

4.1 A Payout Rule With Unique Subgame Perfect Nash Equilibrium

As before, our payouts will come in two varieties. Every time a miner mines a block on the attacking chain (as opposed to the defending chain), they lose their block reward R that would have been paid out on the defending chain. We therefore provide them a payout of R *via the defending chain* every time they mine an attacking block. This means that, no matter the eventual state of the attack, their block-reward payouts are identical across all of their strategy options, and their net rewards therefore follow those of the attacking game described above up to a constant. To incentivize miners to participate in the attack, it therefore suffices to provide a payout rule which enforces a unique Nash equilibrium of participation in said game. From an intuitive point of view, we hope to create a payment rule which causes the attack to have a very high chance of succeeding regardless of the behavior of any individual miner. This, in turn, should allow us to pay each individual miner very little, as their marginal loss as a result of participation will be low.

We first define a pseudo-value function

$$w^{\max}_{t,l} = \begin{cases} l = 0: & 0 \\ t = T, l > 0: & v\gamma R \\ t < T, l > 0: & g_{\mathrm{Def}} w^{\max}_{t+1,l+1} + (1 - g_{\mathrm{Def}}) w^{\max}_{t+1,l-1} + \gamma(w^{\max}_{t+1,l+1} - w^{\max}_{t+1,l-1}) \end{cases}$$

This is motivated by the following theorem:

▶ **Theorem 1.** *Fix a payout scheme where the miner who mines block t from the initial state $(t-1, l)$ receives (if they mined the block on the attacking chain) $c_{t-1,l} = (w^{\max}_{t,l+1} - w^{\max}_{t,l-1})$. This scheme admits a unique subgame perfect Nash equilibrium in the mining game described earlier, under which all non-committed miners participate in the attack on every block.*

Proof. We define the value function for miner m as

$$w_{t,l}(m) = \begin{cases} l = 0: & 0 \\ t = T, l > 0: & v\frac{\phi(m)}{\Phi}R \\ t < T, l > 0: & g_{\text{Def}}w_{t+1,l+1} + (1 - g_{\text{Def}})w_{t+1,l-1} + \frac{\phi(m)}{\Phi}c_{t,l} \end{cases}$$

We induct backwards from $t = T$ on the joint hypotheses that:

1. $w_{t,l}(m)$ is equal to the expectation of the sum of attack payouts and time-discounted post-attack mining rewards for a miner m in the non-committed group playing the subgame perfect Nash equilibrium strategy.
2. Any non-committed miner selected at time t is incentivized to participate in the attack at step t given the above payout.

The first claim is trivial at $t = T$, as the attack's success/failure is determined at the end of step T, and our choice for $w_{T,l}$ reflects the expected future rewards under those circumstances. The second claim at $t = T$ follows fairly simply: if their decision would determine attack success, they are paid $v\gamma R > v\frac{\phi(m)}{\Phi}R$ (their expected loss from attack success), and if not, they are paid nothing (for uniqueness, assume a very small payment in this case).

Given that the hypotheses hold for $t > t_0$, we may show that hypothesis 1 holds for $t = t_0$: observe that as every non-committed miner will attack on the next step, the state (t_0, l) has chance g_{Def} of evolving to $(t_0 + 1, l + 1)$ and chance $(1 - g_{\text{Def}})$ of evolving to $(t_0 + 1, l - 1)$. Moreover, the chance that miner m will be selected to mine the next block (and therefore reap an additional reward of $c_{t_0-1,l} = (w^{\max}_{t_0,l+1} - w^{\max}_{t_0,l-1})$) is $\frac{\phi(m)}{\Phi}$.

It remains to show hypothesis 2 for $t = t_0$ given hypothesis 1 for $t \geq t_0$ and hypothesis 2 for $t > t_0$. We may observe that by hypothesis 1, the loss incurred for participating on step t_0 is $(w_{t_0,l+1}(m) - w_{t_0,l-1}(m))$. It therefore suffices to demonstrate that this is less than $w^{\max}_{t_0,l+1} - w^{\max}_{t_0,l-1}$. $w_{t,l}(m)$ is in fact positive-linear in $\frac{\phi(m)}{\Phi}$, and so as it is increasing in l, $(w_{t_0,l+1}(m) - w_{t_0,l-1}(m))$ is positive-linear in the same. At $\frac{\phi(m)}{\Phi} = \gamma$, we have $w_{t,l}(m) = w^{\max}_{t,l}$ and so $(w_{t_0,l+1}(m) - w_{t_0,l-1}(m)) = (w^{\max}_{t_0,l+1} - w^{\max}_{t_0,l-1})$; therefore, for $\frac{\phi(m)}{\Phi} \leq \gamma$, we have $(w_{t_0,l+1}(m) - w_{t_0,l-1}(m)) \leq (w^{\max}_{t_0,l+1} - w^{\max}_{t_0,l-1})$ as desired. ◀

The attacker must be able to credibly commit to their payout scheme to make the reward scheme accurate (and therefore for the equilibrium to hold). This is reasonable – presuming that their payout for attack failure is at least $v\gamma R$ plus the sum of their payouts (which we will see is small), they are incentivized to keep paying under the Nash equilibrium regardless of the game state (as their reward is inverse to that of a miner with power γv). This may also be generalized to arbitrary behavior, assuming that the attacker and the participant miners can agree on an expectation of future miner behavior.

4.2 Success Likelihood, Expected Attack Length, and Expected Attack Cost

We assume $g_{\mathrm{Def}} + \gamma < \frac{1}{2}$. Attack success probability is at least the probability that, had the attack been allowed to continue for T steps whether or not l_t became 0, l_T would have been ≤ 0. We know that l_t decreases with probability $1 - g_{\mathrm{Def}}$ and increases with probability g_{Def}, so after T steps, it has expectation $l_0 - T(1 - 2g_{\mathrm{Def}})$ (assuming counterfactually that we continued the attack after $l_t = 0$). Then for $T > \frac{l_0}{1-2g_{\mathrm{Def}}}$, Hoeffding's inequality gives that the probability that $l_T > 0$ is

$$\leq \exp\left[-\frac{\frac{1}{2}\left(l_0 - T(1-2g_{\mathrm{Def}})\right)^2}{T}\right]$$

Expected time to attack conclusion is bounded by the expected stopping time of the biased random walk described above, equal to $\frac{l_0}{1-2g_{\mathrm{Def}}}$.

We now prove a generic bound on $c_{t-1,l}$. First, let $\left(X^{t,l}\right)_{t,l}$ be a (t,l)-indexed collection of random walks on the integers s.t. $X^{t,l}$ starts at time t at position l, goes to time T, and increases/decreases at each step with probabilities $(g_{\mathrm{Def}} + \gamma)$, $(1 - g_{\mathrm{Def}} - \gamma)$ respectively. For $i \geq t$, let $X_i^{t,l}$ be the position of $X^{t,l}$ at time i. The choice of the step probabilities gives us that, by its definition, $w_{t,l}^{\max} = v\gamma R \Pr\left[\min_{i\in\{t,\ldots,T\}} X_i^{t,l} \leq 0\right]$. Now by coupling the walks $X^{t,l+1}$ and $X^{t,l-1}$ so that one increases on step i iff the other does, we have

$$\begin{aligned} w_{t,l+1}^{\max} - w_{t,l-1}^{\max} &= v\gamma R \Pr\left[\min_{i\in\{t,\ldots,T\}} X_i^{t,l+1} \in \{1,2\}\right] \\ &= v\gamma R \sum_{\tau=t}^{T} \Pr\left[X_\tau^{t,l+1} \in \{1,2\}\right] \Pr\left[\tau = \operatorname*{arg\,min}_{i\in\{t,T\}} X_i^{t,l+1} | X_\tau^{t,l+1} \in \{1,2\}\right] \end{aligned}$$

Each of the terms in the sum may be bounded. Proofs of these results are contained in the appendix.

▶ **Lemma 2.**

$$\Pr\left[X_\tau^{t,l+1} \in \{1,2\}\right] \leq \frac{1}{\sqrt{\tau - t}} \frac{(1 - g_{\mathrm{Def}} - \gamma)^{5/2}}{\sqrt{2\pi}\,(g_{\mathrm{Def}} + \gamma)^{7/2}}$$

▶ **Lemma 3.**

$$\Pr\left[\tau = \operatorname*{arg\,min}_{i\in\{t,T\}} X_i^{t,l+1} | X_\tau^{t,l+1} \in \{1,2\}\right] \leq e^{-\frac{1}{2}(T-\tau)(1-2g_{\mathrm{Def}}-2\gamma)^2}$$

We also require a technical lemma:

▶ **Lemma 4.** *Let* $0 < a \leq 1$. *Then*

$$\sum_{i=t}^{T} \frac{e^{-a(T-i)}}{\max(\sqrt{i-t}, 1)} \leq \min\left(\frac{2}{\sqrt{T + 1 - t - 2\frac{\ln(1+\sqrt{T+1-t})}{a}}}, 1 + \frac{1}{1 - e^{-a}}\right)$$

Combined, these yield a straightforward bound on worst-case total attack cost:

▶ **Theorem 5.** *The worst-case cost of the* $w^{\max}$*-attack is bounded by*

$$TR + \sum_{t=1}^{T} c_{t-1,l_t} \leq TR + v\gamma R\left[\frac{4\ln(1+\sqrt{T})}{(1 - 2g_{\mathrm{Def}} - 2\gamma)} + \sqrt{\frac{2}{\pi}}\frac{(1 - g_{\mathrm{Def}} - \gamma)^{5/2}}{(g_{\mathrm{Def}} + \gamma)^{7/2}} 2\sqrt{T}\right]$$

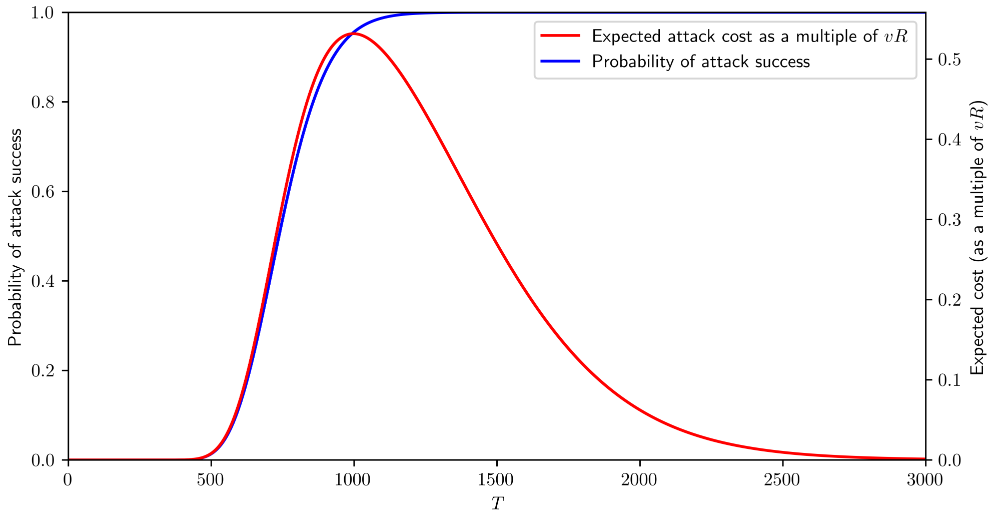


Figure 1 Probability of attack success and expected attack cost (excluding per-block payouts of R) for an attack with given T and $l_0 = 150, \gamma = 0.05, g_{\text{Def}} = 0.4$.

We also have a much lower bound on expected attack cost.

▶ **Theorem 6.** *The expected cost of the $w^{\max}$-attack is bounded by*

$$R\left(\frac{l_0}{2} + \frac{1}{2}\frac{l_0}{1 - 2g_{\text{Def}}}\right) + v\gamma RTe^{-\frac{(1-2g_{\text{Def}}-2\gamma)^2 T/2 - (1-2g_{\text{Def}})l_0}{2}}$$

Critically, for sufficiently large T, we observe exponential decay in the expected attack cost beyond the per-block payout.

These bounds are far from tight. A graph of expected attack cost (excluding the per-block payouts of R) and attack success likelihood in terms of T for an attack against $l_0 = 150$ is included in Figure 1. The attack cost peaks when success is uncertain, as this is where participation/refusal has the strongest effect, and falls off rapidly as attack success becomes certain.

Two remaining points about the asymptotics are of interest. First, as l_0 increases, both attack success chance and attack cost converge more quickly in terms of $\frac{T}{l_0}$, as the expected result concentrates better. In particular, the premium over the naive cost of mining energy required to perform a range-l_0 attack in which $1 - g_{\text{Def}}$ of mining power is participating is logarithmic in l_0, not linear. Second, due to the exponential term, γ does not need to be low to make the attack cost very low. For instance, $\gamma = .2, g_{\text{Def}} = .25$ yields quite feasible attacks for $\frac{T}{l_0}$ substantially greater than $\frac{1-2g_{\text{Def}}}{(1-2g_{\text{Def}}-2\gamma)^2}$. However, only a mild amount of successful collusion (two miners of size .15, for instance) is needed to prevent this attack.

5 Relative Immunity of Proof-of-Stake: Ethereum

Ethereum contains a significant defense against attacks of this type: slashing [5]. As Ethereum severely penalizes stakers who validate on two incompatible chain branches, the internalized penalty for attack participation as compared to attack non-participation may be as high as the Budish payout if the attack is not expected to reduce the price of Ethereum. If attackers will lose their stake, the expected future payoff matrix is

	Attack succeeds	Attack fails
Participate	0	0
Refuse	0	$\frac{\phi(m)}{\Phi}vR$

This implies that short-range attacks (those in which attackers must possess vulnerable stake on the true chain) will cost close to the Budish estimate. Ethereum also has weak subjectivity: the property that any agent entering the network with access to a sufficiently recent honest network state can independently determine the present honest state [4]. Assuming access to such states for entrants, this renders long-range attack impossible, so decentralization does not meaningfully reduce Ethereum attack cost.

6 Practical Economic Considerations

Attackers face two primary logistical difficulties: estimating the total power of participants and coordinating the attack. For PoW blockchains, both of these are ameliorated by setting up a specialized mining pool (with similar structure to that of [2], albeit different intent). Such a pool could function normally until attack time, at which point it could begin sending work units for the attack chain, instead of the consensus chain. This approach has important secondary advantages: mining pools are commonly understood and easy to work with, and by offering low pool fees (or even paying pool participants slightly more than they mine, as suggested by [2] for a different purpose), switchover in advance of the attack could be incentivized while negligibly increasing attack cost. Stratum v2 supports header-only mining, allowing miners to attempt to mine an externally specified block, which should allow easy coordination of attack mining. The remaining coordination difficulty is in payout – corruption payouts cannot flow through the original chain due to likelihood of devaluation. Assuming that another cryptocurrency (for instance, a PoS-based one) is expected to be unaffected by the attack, it could be used as a payment medium; otherwise, a classical medium would be needed.

Regarding pool mining, we also note that the attack proposed in Section 4 can be enhanced to provide lower payout variability with small increase in cost: in addition to the payouts given to participants who mine blocks, duplicate the defending-chain payout and split it proportionate to mining power across the pool. This guarantees participants the same consistency of payout as they would have received had they been participating in a large mining pool, while keeping cost similar, and may incentivize risk-averse/small miners to participate.

7 Attacker Incentives and Dangers

We first estimate the cost of a medium-range majority attack on Bitcoin. Distribution of control of mining capacity is a closely guarded secret, but mining power appears to be significantly spatially and internationally distributed [13]. Assume that miners representing 60% of mining power will participate if we set $\gamma = 1/20$ (i.e. $g_{\mathrm{Def}} = 0.4$). Consider an attack with a backwards range of $l_0 = 150$ blocks (about one day, for Bitcoin). Solving the recursion explicitly, expected attack cost for $T = 2500$ is $< (0.01v + 450)R$, and probability of success $\geq 1 - 10^{-12}$. Assuming $v = 158000$ (the number of Bitcoin blocks in a little over 3 years), expected attack cost is $\leq 1942R$, or a little under 12.2k BTC/310M USD at time of writing. Maximum potential attack cost is substantially higher – $2.38vR$, or about 2.35M BTC/60B USD – but given that attack cost concentrates with exponential falloff, this may be insurable. γ and g_{Def} may also be substantially overestimated here, leading to inflated cost bounds

– if we take $\gamma = 0.03, g_{\text{Def}} = 0.2$, maximum payout is less than 820k BTC/21B USD and expected payout is just over 1293 BTC/33M USD for a $T = 400$ attack (with attack success chance $\geq 1 - 10^{-7}$).

This attack is cheap enough to be profitable, and therefore dangerous. Open Bitcoin options volume has consistently held above \$5 billion USD for four years, and has been substantially higher during peaks. Daily trading volume of BTC has stayed well above 200000 BTC for a similar period, and most BTC options are relatively short-term. Attackers able to successfully trade and liquidate options at these scales could make substantial profit.

One example of an attack strategy is as follows. First, buy B "covering" units of BTC, and short-sell a separate B units of BTC. Sell the covering units, and attack to revert that transfer. Provided that the attack succeeds, the recouped covering bitcoin may be used to cover the short position, and as its initial buy cost and sale profit are approximately equal, the net profit of the attack is equal to the gross income from the short sale. In the case of attack failure, the buy and sell costs of the covering units cancel, and cost is equal to the cost of covering the short positions minus the profit from selling them (i.e., it is as if the attackers had simply shorted BTC), less transaction costs from the purchase and sale. Even in this case, the attack may substantially devalue BTC, creating some profit from the short position.

If blockchain technologies become more mainstream, non-currency-oriented attacks will become increasingly appealing. When substantial value may be placed on non-currency attributes of the chain (such as smart contract or DAO state, ownership of a NFT, etc.) in unpredictable ways, many non-obvious attack incentives will exist. This also makes attack attribution more difficult.

In addition to potentially being profitable, a majority attack may significantly if not completely devalue the attacked currency, and possibly others as well. The economic impact of this could be substantial. At time of writing, the global market cap of cryptocurrencies stands at about \$1 trillion USD, and various state or non-state actors may stand to gain substantially from a crash precipitated by an attack.

8 Economic Prevention and Mitigation

On a protocol level, this attack is equivalent to a 51% attack, and the same impossibility results apply to protocol-level defense. On a non-protocol level, [2] discusses several prevention measures. We discuss the most relevant of these, and some others, here.

8.1 Coalitions

In response to an incipient attack, a coalition of miners may agree on a mutual behavior contract designed to disincentivize attack participation. In particular, miners could commit to non-subgame-perfect equilibrium strategies, such as defending until a certain threshold of miners have been shown to participate in the attack, and then attacking thereafter, which stymie the given payout rule. However, credible commitment is difficult for miners, and such Nash equilibria are not robust to the attacker designing new payout schemes in response. We conjecture that given any such commitment scheme, the attacker can design a bribery system with expected payout and failure probability bounded by those of the subgame perfect Nash equilibrium and corresponding payout rule. For instance, in the above example, the attacker could increase payouts to miners until the threshold has been passed, and then remove them once it has.

Coalitions with leaders, such as some mining pools, present a separate danger to blockchains: if coalition mining power is controlled by a player whose expected future payout from attack failure is lower than vR, the controller may be incentivized to make the coalition participate even if it falls above the power threshold. For instance, if the controller of some mining pool receives 1% of the profits generated, they will be incentivized to cause the pool to participate if its power is $\leq 100\gamma\Phi$.

8.2 Social Consensus as Deterrence/Mitigation

If a sufficiently broad set of participants wishes, they may fork or otherwise alter the blockchain after an attack in an attempt to reverse the effects of the attack. This is unlikely to prevent profit-taking, as it must be done rapidly enough to fix all manipulated transactions before they can be used for profit. For instance, in the attack proposed in the previous section, the sale of B units of BTC must be fixed as canonical before it can be sold again in the attack chain. Moreover, this has obvious disadvantages for regular use, and does not prevent repeated sabotage attacks [3], which would effectively make the blockchain unusable (which can itself be profitable for an attacker).

8.3 Counterattacks and the Model of Moroz et al.

Moroz et al. [10] argue that in many cases, the threat of counterattacks renders double-spend attacks unprofitable. They analyze a model in which a single large transaction is in contention between the sender (the attacker) and the receiver (the defender), with the former wishing to establish a heaviest chain not including the transaction and the latter wishing the opposite. Each party is allowed to purchase mining power on an open market in order to attempt majority attacks to shift consensus, and take turns doing so. Moroz et al. find that under certain conditions, the only equilibrium strategy is to not attack in the first place.

However, their model (and therefore results) are inapplicable in our case, as both parties may have external rewards which do *not* depend on the final status of the transaction. The success of one or more attacks will almost certainly substantially reduce the value of the attacked cryptocurrency, independent of the final state of the blockchain. This itself may be a major source of utility for the attacker (see Section 7) and disutility for the defender (the defender will recoup coins of lesser value if they are successful, and if they hold additional units of the currency, those two will be devalued by attacks).

When this assumption is changed, the no-attack equilibrium established by Moroz et al. does not hold, and attack with no response is equilibrium in many practical cases. For instance, if a single successful attack will massively devalue a currency, but the attacker can make profit from this event, the attack is incentivized *even if it will be counterattacked*, and a counterattack is generally not incentivized.

8.4 Countercorruption

As noted in [2], miners above the threshold, targets of a large double-spend, and other entities with stake in the success of blockchains may be incentivized to attempt to bribe miners to *not* participate. While this may be theoretically sufficient in some cases, it is undesirable for a variety of reasons discussed in [2].

8.5 Extra Confirmations

Historically, requiring more confirmations has been seen as a natural way to secure transactions. However, the attack cost for even long-range attacks is relatively low, and the splitting tactic noted by [2] likely allows evasion unless many transactions require extremely long confirmation periods.

9 Non-Economic Prevention and Mitigation

We see that there exist potential large-scale corruptive majority attacks which are incentive-compatible for all participants and highly profitable for the attacker, and that economic forces are insufficient to disincentivize attack initiation or participation. However, this does not preclude non-economic forces preventing these attacks in practice. We briefly discuss three potential avenues of prevention.

9.1 Social Consensus

If a power-weighted majority of miners refuse to participate, the attack will be stymied [2]. However, for this to work, we require a broad coalition of miners to act against their own self-interest for philosophical or other non-economic reasons. Miners have regularly behaved selfishly in contravention of protocol [15], so confidence that they will behave selflessly in this case seems misplaced. For instance, miners did not leave F2Pool in large numbers despite its timestamp manipulation, implying that miners are willing to participate in activity with negative network consequences for mildly increased personal profit.

9.2 Force

Actors subject to the jurisdiction of a force (legal or otherwise) able to both detect attack organizers/participants and impose sufficient punishments against them are unlikely to perform/abet attacks. Most immediately, various state actors are likely to take a dim view of such an attack. In the United States, for instance, any attempt to organize such an attack would likely constitute securities market manipulation, and while the legal system is woefully unequipped to deal with the consequences of a double-spend attack, organizing (and perhaps merely participating in) one would likely incur substantial civil and criminal liability. This may be enough to dissuade most actors from performing a majority attack, but it will not be enough to disincentivize those who operate outside the bounds of the law and/or with other motivations. Even individuals subject to regulatory jurisdiction may be insufficiently deterred if the attack is conducted through privacy-preserving tools, as attribution in such a case might be difficult.

9.3 Non-Profitability

One of the incentives to carry out such an attack is profitability. Substantially reducing the viability of profiting off of the collapse of a cryptocurrency could reduce or eliminate the profit motive for majority attacks if its price is expected to collapse by the time the heaviest chain is revised.

10 Conclusion: What *Do* We Trust?

We clarify the analysis of miner incentives surrounding majority attacks, showing that the cost to bribe miners into attacking may be far less than previously believed. The cost is low enough that a corruptive majority attack could be profitable if combined with an appropriate strategy combining shorting and doublespending. Moreover, the value of the attack scales with the value of the blockchain and its associated assets/activity, and the cost of the attack decreases as the miner pool becomes more diverse and incentive-driven, implying that the danger will likely continue to increase as cryptocurrencies grow. Mitigating these attacks through mechanism design is nearly impossible, as doing so would require a method to either force small miners to internalize large social costs or prevent the attacker from profit-taking.

It is unclear whether these attacks can be prevented by non-economic factors. Further work is needed to determine the likelihood of attack in real-world contexts, but such analysis will necessarily be somewhat speculative. Even if these forces can prevent attacks in practice, the advantages of using distributed systems over more classical ledgers become substantially less clear when trust in their stability rests on widespread altruism, institutional force, or other non-economically-motivated behavior. Further analysis of the non-economic forces that may prevent majority attacks, and their implications for the usefulness and viability of blockchain technologies, is warranted.

References

1 Nick Arnosti and S Matthew Weinberg. Bitcoin: A natural oligopoly. *Management Science*, 68(7):4755–4771, 2022.

2 Joseph Bonneau. Why buy when you can rent? bribery attacks on bitcoin-style consensus. In *Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers 20*, pages 19–26. Springer, 2016.

3 Eric Budish. The economic limits of bitcoin and the blockchain. Technical report, National Bureau of Economic Research, 2018.

4 Vitalik Buterin. Proof of stake: How i learned to love weak subjectivity, November 2014. URL: `https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity`.

5 Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

6 Piotr Faliszewski, Jörg Rothe, and Hervé Moulin. Control and bribery in voting, 2016.

7 Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gaži, Sarah Meiklejohn, and Edgar Weippl. Sok: Algorithmic incentive manipulation attacks on permissionless pow cryptocurrencies. In *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25*, pages 507–532. Springer, 2021.

8 Kevin Liao and Jonathan Katz. Incentivizing blockchain forks via whale transactions. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*, pages 264–279. Springer, 2017.

9 Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In *Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers 22*, pages 3–18. Springer, 2019.

10 Daniel J Moroz, Daniel J Aronoff, Neha Narula, and David C Parkes. Double-spend counterattacks: Threat of retaliation in proof-of-work systems. *arXiv preprint arXiv:2002.10736*, 2020.

11 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

12 Savva Shanaev, Arina Shuraeva, Mikhail Vasenin, and Maksim Kuznetsov. Cryptocurrency value and 51% attacks: evidence from event studies. *The Journal of Alternative Investments*, 22(3):65–77, 2019.

13 Wei Sun, Haitao Jin, Fengjun Jin, Lingming Kong, Yihao Peng, and Zhengjun Dai. Spatial analysis of global bitcoin mining. *Scientific Reports*, 12(1):1–12, 2022.

14 Lirong Xia. The impact of a coalition: Assessing the likelihood of voter influence in large elections. In *Proceedings of the 24th ACM Conference on Economics and Computation*, pages 1156–1156, 2023.

15 Aviv Yaish, Gilad Stern, and Aviv Zohar. Uncle maker:(time) stamping out the competition in ethereum. *Cryptology ePrint Archive*, 2022.

A Notation Reference

For convenience, we define the notations that appear in this paper here:

- M: the set of miners.
- Φ: the total mining power of all miners.
- $\phi : M \to [0, \Phi]$: the function mapping miners to their mining powers.
- T: the duration, in blocks, of an attack.
- R: the (expected) value of mining a block under the stable condition of the blockchain.
- v: a multiplier such that any miner m who expects per-block reward $\frac{\phi(m)}{\Phi}R$ has time-discounted total future profits $\leq v\frac{\phi(m)}{\Phi}R$. That this exists (and may be reasonably bounded) follows from time-discounting, reduction in rewards over time, and hardware improvement/death.
- $f\left(\frac{\phi(m)}{\Phi}\right)$: an upper bound on miner m's estimate of the probability that their participation/non-participation in the attack will determine whether it succeeds.
- $f_{\max}$: a threshold on f. We design an attack which will incentivize participation by miners with $f\left(\frac{\phi(m)}{\Phi}\right) \leq f_{\max}$.
- γ: a threshold on $\frac{\phi(m)}{\Phi}$ with similar purpose to the above.
- g_{Def}: the fraction of total power held by honest miners.

B Proofs of Results in Section 4

Lemma 2

Proof.

$$\begin{aligned}
&\Pr\left[X_\tau^{t,l+1} \in \{1,2\}\right] \\
=&\binom{\tau-t}{\frac{\tau-t+l}{2}}(1-g_{\text{Def}}-\gamma)^{\frac{\tau-t+l}{2}}(g_{\text{Def}}+\gamma)^{\frac{\tau-t-l}{2}} \\
&\quad+\binom{\tau-t}{\frac{\tau-t+l-1}{2}}(1-g_{\text{Def}}-\gamma)^{\frac{\tau-t+l-1}{2}}(g_{\text{Def}}+\gamma)^{\frac{\tau-t-l+1}{2}}
\end{aligned}$$

where the binomial coefficient $\binom{n}{k}$ is taken to be 0 if k is fractional. One term will always be 0, and so letting $m = \tau - t$, we may bound this by

$$\sup_{l\in\{0,1,\ldots,m\}} \binom{m}{\frac{m+l}{2}}(1-g_{\mathrm{Def}}-\gamma)^{\frac{m+l}{2}}(g_{\mathrm{Def}}+\gamma)^{\frac{m-l}{2}}$$

Note that this is effectively only over l of the same parity as $\tau - t$.

Let $K_l = \binom{m}{\frac{m+l}{2}}(1-g_{\mathrm{Def}}-\gamma)^{\frac{m+l}{2}}(g_{\mathrm{Def}}+\gamma)^{\frac{m-l}{2}}$. This admits a smooth extension to $l \in [0, \tau-t]$ via the gamma function. We may bound this extension (via Stirling approximation) by

$$K_l \le \sup_{l\in[0,\tau-t]} \sqrt{\frac{m}{2\pi\frac{m+l}{2}\frac{m-l}{2}}}\frac{(m)^m}{\left(\frac{m+l}{2}\right)^{\frac{m+l}{2}}\left(\frac{m-l}{2}\right)^{\frac{m-l}{2}}}(1-g_{\mathrm{Def}}-\gamma)^{\frac{m+l}{2}}(g_{\mathrm{Def}}+\gamma)^{\frac{m-l}{2}} \quad \le$$

We observe that for integer l of parity $\tau - t$,

$$\begin{aligned}\frac{K_{l+2}}{K_l} &= \frac{\frac{m!}{(\frac{m-(l+2)}{2})!(\frac{m+(l+2)}{2})!}}{\frac{m!}{(\frac{m-l}{2})!(\frac{m+l}{2})!}}\frac{1-g_{\mathrm{Def}}-\gamma}{g_{\mathrm{Def}}+\gamma}\\ &= \frac{m-l}{m+l+2}\frac{1-g_{\mathrm{Def}}-\gamma}{g_{\mathrm{Def}}+\gamma}\end{aligned}$$

In particular, K_l is maximized across integers at the lowest l of parity $\tau - t$ s.t. the above ratio is ≤ 1 (as this ratio is decreasing in l). Solving $\frac{m-l}{m+l+2}\frac{1-g_{\mathrm{Def}}-\gamma}{g_{\mathrm{Def}}+\gamma} = l$ gives $l = m(1-2(g_{\mathrm{Def}}+\gamma))-2(g_{\mathrm{Def}}+\gamma)$. In particular, the integer $l^{\max}$ maximizing K_l is somewhere between $m(1-2(g_{\mathrm{Def}}+\gamma))-3$ and $m(1-2(g_{\mathrm{Def}}+\gamma))$. Plugging $l^* = m(1-2(g_{\mathrm{Def}}+\gamma))$ into our bound on K_l gives

$$\frac{1}{2\sqrt{m}}\sqrt{\frac{1}{2(g_{\mathrm{Def}}+\gamma)(1-2(g_{\mathrm{Def}}+\gamma))}}$$

Moreover, we may note that

$$\begin{aligned}\frac{K_{l^*-c}}{K_{l^*}} &= \frac{\Gamma\left(\frac{m+l^*}{2}\right)/\Gamma\left(\frac{m+l^*-c}{2}\right)}{\Gamma\left(\frac{m-l^*+c}{2}\right)/\Gamma\left(\frac{m-l^*}{2}\right)}(1-g_{\mathrm{Def}}-\gamma)^{c/2}(g_{\mathrm{Def}}-\gamma)^{-c/2}\\ &\le \left(\frac{(m+l^*)(1-g_{\mathrm{Def}}-\gamma)}{(m-l^*)(g_{\mathrm{Def}}+\gamma)}\right)^{c/2}\\ &= \left(\frac{1-g_{\mathrm{Def}}-\gamma}{g_{\mathrm{Def}}+\gamma}\right)^{c}\end{aligned}$$

Then letting $l^{\max} \in [l^*-3, l^*]$ be the maximizer of K_l, we obtain that

$$\sup_l K_l = K_{l^{\max}} \le \frac{1}{\sqrt{2\pi m}}\sqrt{\frac{1}{(g_{\mathrm{Def}}+\gamma)(1-g_{\mathrm{Def}}-\gamma)}}\left(\frac{1-g_{\mathrm{Def}}-\gamma}{g_{\mathrm{Def}}+\gamma}\right)^3 \qquad ◀$$

Lemma 3

Proof. It suffices to observe that

$$\begin{aligned}\Pr\left[\tau = \arg\min_{i\in\{t,T\}} X_i^{t,l+1} | X_\tau^{t,l+1} \in \{1,2\}\right] &\leq \Pr\left[\tau = \arg\min_{i\in\{\tau,T\}} X_i^{t,l+1} | X_\tau^{t,l+1} \in \{1,2\}\right]\\ &= \Pr\left[\tau = \arg\min_{i\in\{\tau,T\}} X_i^{\tau,X_\tau^{t,l+1}}\right]\\ &\leq \Pr\left[X_T^{\tau,X_\tau^{t,l+1}} \geq X_\tau^{\tau,X_\tau^{t,l+1}}\right]\\ &\leq e^{-\frac{1}{2}(T-\tau)(1-2g_{\mathrm{Def}}-2\gamma)^2}\end{aligned}$$

by Hoeffding's inequality applied to $X_T^{\tau,X_\tau^{t,l+1}} - X_\tau^{\tau,X_\tau^{t,l+1}}$, which is by definition a sum of $T-\tau$ random variables which are independently -1 with probability $1-g_{\mathrm{Def}}-\gamma$ and 1 with probability $g_{\mathrm{Def}}+\gamma$. ◀

Lemma 4

Proof. $T = t$ is trivial. Otherwise, fix $t < k \leq T$. Then

$$\begin{aligned}\sum_{i=t}^{T} \frac{e^{-a(T-i)}}{\max(\sqrt{i-t},1)} &\leq \sum_{i=t}^{k-1} \frac{e^{-a(T-i)}}{\max(\sqrt{i-t},1)} + \sum_{i=k}^{T} \frac{e^{-a(T-i)}}{\max(\sqrt{i-t},1)}\\ &\leq \sum_{i=t}^{k-1} \frac{e^{-a(T-(k-1))}}{\max(\sqrt{i-t},1)} + \sum_{i=k}^{T} \frac{e^{-a(T-i)}}{\sqrt{k-t}}\\ &\leq \left(1+\sqrt{k-1-t}\right) e^{-a(T-(k-1))} + \frac{1}{\sqrt{k-t}}\frac{1}{1-e^{-a}}\end{aligned}$$

Setting $k = \max\left(\lceil T - 2\frac{\ln(1+\sqrt{T+1-t})}{a} + 1\rceil, t+1\right)$ yields that the above is

$$\begin{aligned}&\leq \min\left(\frac{1}{1+\sqrt{T+1-t}} + \frac{1}{\sqrt{T+1-t-2\frac{\ln(1+\sqrt{T+1-t})}{a}}}, 1+\frac{1}{1-e^{-a}}\right)\\ &\leq \min\left(\frac{2}{\sqrt{T+1-t-2\frac{\ln(1+\sqrt{T+1-t})}{a}}}, 1+\frac{1}{1-e^{-a}}\right)\end{aligned}$$ ◀

Theorem 5

Proof. We have

$$\begin{aligned}c_{t-1,l} &= w_{t,l+1}^{\max} - w_{t,l-1}^{\max}\\ &= v\gamma R \sum_{\tau=t}^{T} \Pr\left[X_\tau^{t,l+1} \in \{1,2\}\right] \Pr\left[\tau = \arg\min_{i\in\{t,T\}} X_i^{t,l+1} | X_\tau^{t,l+1} \in \{1,2\}\right]\\ &\leq v\gamma R \frac{(1-g_{\mathrm{Def}}-\gamma)^{5/2}}{\sqrt{2\pi}\,(g_{\mathrm{Def}}+\gamma)^{7/2}} \sum_{\tau=t}^{T} \frac{e^{-\frac{1}{2}(T-\tau)(1-2g_{\mathrm{Def}}-2\gamma)^2}}{\max\left(\sqrt{\tau-t},1\right)}\end{aligned}$$

by Lemmas 2 and 3. Applying Lemma 4 with $a = \frac{(1-2g_{\mathrm{Def}}-2\gamma)^2}{2}$, combined with the trivial bound $c_{t-1,l} \leq w_{t,l+1}^{\max} \leq v\gamma R$ yields

$$\begin{aligned}
c_{t-1,l} &\le v\gamma R \min\left(\frac{(1-g_{\text{Def}}-\gamma)^{5/2}}{\sqrt{2\pi}\,(g_{\text{Def}}+\gamma)^{7/2}}\frac{2}{\sqrt{T+1-t-4\frac{\ln(1+\sqrt{T+1-t})}{(1-2g_{\text{Def}}-2\gamma)^2}}}, 1\right)\\
&\le v\gamma R \min\left(\frac{(1-g_{\text{Def}}-\gamma)^{5/2}}{\sqrt{2\pi}\,(g_{\text{Def}}+\gamma)^{7/2}}\frac{2}{\sqrt{T+1-t-4\frac{\ln(1+\sqrt{T})}{(1-2g_{\text{Def}}-2\gamma)^2}}}, 1\right)
\end{aligned}$$

Then the result follows directly from partitioning the indices at $T+1-4\frac{\ln(1+\sqrt{T})}{(1-2g_{\text{Def}}-2\gamma)^2}$. ◀

Theorem 6

Proof. Fix two walks X and Y on the integers starting at l_0. At each step, let X increment with probability g_{Def} and decrement with probability $1-g_{\text{Def}}$, and let Y increment with probability $g_{\text{Def}}+\gamma$ and decrement with probability $1-g_{\text{Def}}-\gamma$. X will correspond to game state (until hitting 0 – i.e. the distributions of l_t and X_t are identical over positive integers), and Y_t is a virtual walk corresponding to the $w^{\max}$ recurrence.

Fix $k > l_0$. By Hoeffding's inequality on X_t, the probability that $l_t \ge k$ for given t is $\le e^{-\frac{(k+(1-2g_{\text{Def}})t)^2}{2t}} \le e^{-(1-2g_{\text{Def}})(\frac{(1-2g_{\text{Def}})t}{2}+k)}$. Taking a union bound across all t, the probability that there exists a timestep t with $l_t \ge k$ is

$$\begin{aligned}
&\le \sum_{i=1}^{T} e^{-(1-2g_{\text{Def}})(\frac{(1-2g_{\text{Def}})t}{2}+k)}\\
&< \sum_{t=1}^{\infty} e^{-(1-2g_{\text{Def}})(\frac{(1-2g_{\text{Def}})t}{2}+k)}\\
&< \frac{e^{-(1-2g_{\text{Def}})k}}{1-e^{-\frac{(1-2g_{\text{Def}})^2}{2}}}
\end{aligned}$$

We may observe by its recursion that

$$\begin{aligned}
w^{\max}_{t,l} &= v\gamma R \Pr[\min_{i\in[T]} Y_i > 0 | Y_t \le l]\\
&\le v\gamma R \Pr[Y_T > 0 | Y_t \le l]\\
&\le \begin{cases} (1-2g_{\text{Def}}-\gamma)(T-t) < l: & v\gamma R\\ (1-2g_{\text{Def}}-\gamma)(T-t) \ge l: & v\gamma R e^{-\frac{(l-(1-2g_{\text{Def}}-\gamma)(T-t))^2}{2(T-t)}} \end{cases}\\
&\le v\gamma R e^{l(1-2g_{\text{Def}}-2\gamma)-\frac{(1-2g_{\text{Def}}-2\gamma)^2(T-t)}{2}}
\end{aligned}$$

where the concentration follows from Hoeffding's inequality. We may also bound the probability that the attack has not succeeded by the start of step t, conditioned on there not existing a timestep t with $l_t \ge k$, as

$$\begin{aligned}
\Pr[X_{t-1} > 0 | \sup_{i\in[T]} X_i < k] &\le \Pr[X_t \ge 0 | \sup_{i\in[T]} X_i < k]\\
&\le \Pr[X_t \ge 0]\\
&\le e^{l_0(1-2g_{\text{Def}})-\frac{(1-2g_{\text{Def}})^2 t}{2}}
\end{aligned}$$

by the same.

Finally, we observe that

$$c_{t-1,X_{t-1}} \leq w^{\max}_{t,X_{t-1}+1} \leq e^{(X_{t-1}+1)(1-2g_{\text{Def}}-2\gamma)-\frac{(1-2g_{\text{Def}}-2\gamma)^2(T-t)}{2}}$$

In particular, conditioned on $\forall i \in [T] : X_i < k$, we have

$$c_{t-1,l}\Pr[X_t \geq 0 | \forall i \in [T] : X_i < k] \leq e^{l_0(1-2g_{\text{Def}})+k(1-2g_{\text{Def}}-2\gamma)-T(1-2g_{\text{Def}}-2\gamma)^2/2}$$

Then we bound expected attack cost C (excluding the per-attacking-block payout) as

$$\begin{aligned}\mathbb{E}[C] &\leq \mathbb{E}[C]\Pr[\exists t \in [T] : X_t \geq k] + \mathbb{E}[C | \forall t \in [T] : X_t < k] \\ &\leq \mathbb{E}[C]\Pr[\exists t \in [T] : X_t \geq k] + \sum_{t=1}^{T} c_{t-1,X_{t-1}<k}\Pr[X_{t-1} > 0 | \sup_{i\in[T]} X_i < k] \\ &\leq v\gamma R\left[T\frac{e^{-(1-2g_{\text{Def}})k}}{1-e^{-\frac{(1-2g_{\text{Def}})^2}{2}}} + Te^{l_0(1-2g_{\text{Def}})+k(1-2g_{\text{Def}}-2\gamma)-T(1-2g_{\text{Def}}-2\gamma)^2/2}\right]\end{aligned}$$

Solving $-(1-2g_{\text{Def}})k = l_0(1-2g_{\text{Def}}) + k(1-2g_{\text{Def}}-2\gamma) - T(1-2g_{\text{Def}}-2\gamma)^2/2$ yields $k = \frac{(1-2g_{\text{Def}}-2\gamma)^2 T/2-(1-2g_{\text{Def}})l_0}{2-4g_{\text{Def}}-2\gamma}$ and therefore total expected cost

$$\begin{aligned}&\leq v\gamma RTe^{-(1-2g_{\text{Def}})\frac{(1-2g_{\text{Def}}-2\gamma)^2 T/2-(1-2g_{\text{Def}})l_0}{2-4g_{\text{Def}}-2\gamma}} \\ &\leq v\gamma RTe^{-\frac{(1-2g_{\text{Def}}-2\gamma)^2 T/2-(1-2g_{\text{Def}})l_0}{2}}\end{aligned}$$

The expected number of attacking blocks mined is bounded by $\frac{l_0}{2} + \frac{1}{2}\frac{l_0}{1-2g_{\text{Def}}}$ (as the biased random walk is expected to last $\frac{l_0}{1-2g_{\text{Def}}}$ steps, and if the attack lasts k blocks, the attackers mine at most $\frac{l+k}{2}$ of them. Then total expected attack cost is bounded by

$$R\left(\frac{l_0}{2} + \frac{1}{2}\frac{l_0}{1-2g_{\text{Def}}}\right) + v\gamma RTe^{\frac{(1-2g_{\text{Def}}-2\gamma)^2 T/2-(1-2g_{\text{Def}})l_0}{2}} \qquad ◀$$

Proofs of Proof-Of-Stake with Sublinear Complexity

Shresth Agrawal
Technische Universität München, Germany

Joachim Neu
Stanford University, CA, USA

Ertem Nusret Tas
Stanford University, CA, USA

Dionysis Zindros
Stanford University, CA, USA

Abstract

Popular Ethereum wallets (like MetaMask) entrust centralized infrastructure providers (*e.g.*, Infura) to run the consensus client logic on their behalf. As a result, these wallets are light-weight and high-performant, but come with security risks. A malicious provider can mislead the wallet by faking payments and balances, or censoring transactions. On the other hand, light clients, which are not in popular use today, allow decentralization, but are concretely inefficient, often with asymptotically *linear* bootstrapping complexity. This poses a dilemma between decentralization and performance.

We design, implement, and evaluate a new proof-of-stake (PoS) *superlight* client with concretely efficient and asymptotically *logarithmic* bootstrapping complexity. Our proofs of proof-of-stake (PoPoS) take the form of a Merkle tree of PoS epochs. The verifier enrolls the provers in a bisection game, in which honest provers are destined to win once an adversarial Merkle tree is challenged at sufficient depth. We provide an implementation for mainnet Ethereum: compared to the state-of-the-art light client construction of Ethereum, our client improves time-to-completion by 9×, communication by 180×, and energy usage by 30× (when bootstrapping after 10 years of consensus execution). As an important additional application, our construction can be used to realize trustless cross-chain bridges, in which the superlight client runs within a smart contract and takes the role of an on-chain verifier. We prove our construction is secure and show how to employ it for other PoS systems such as Cardano (with fully adaptive adversary), Algorand, and Snow White.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Proof-of-stake, blockchain, light client, superlight, bridge, Ethereum

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.14

Related Version *Full Version*: `https://eprint.iacr.org/2022/1642`

Supplementary Material *Software*: `https://github.com/lightclients/poc-superlight-client/tree/master`, archived at `swh:1:dir:0108d316b0418717635ba996b1f6bbcff8fe5b94`

Funding *Joachim Neu*: Supported by the Protocol Labs PhD Fellowship.
Ertem Nusret Tas: Supported by the Stanford Center for Blockchain Research.
Dionysis Zindros: Supported in part by funding from Harmony.

Acknowledgements The authors thank Kostis Karantias for the helpful discussions on bisection games, and Daniel Marin for reading early versions of this paper and providing suggestions. The work of JN was conducted in part while at Paradigm. The work of SA was conducted in part while at Common Prefix.

1 Introduction

If I want to check how much money I have on Ethereum, the most secure way is to run my own full node. Sadly, this requires downloading more than 500 GB of data and can take up to 5 days to sync. Because of this, most Ethereum users today outsource the task of maintaining the latest state to a third-party provider such as Infura or Alchemy. This allows the user to run a lightweight wallet, such as MetaMask[1], on their browser or smartphone.

What can go wrong if Infura is compromised or turns malicious? As per the current wallet design, the wallet will blindly trust the provider and therefore show incorrect data. This is enough to perform a double spend. For example, imagine Bob wants to sell his car to Eve. Regrettably, Eve has compromised Infura. Eve claims that she has paid Bob. Bob checks his MetaMask wallet and sees an incoming and confirmed transaction from Eve. Unfortunately, even though the wallet is non-custodial, this transaction never happened, but is fraudulently reported by Infura to Bob. Since Bob trusts his wallet, and his wallet trusts Infura, he hands over the car keys to Eve. By the time Bob realizes he cannot use this money, Eve has long disappeared with his car in Venice. From the point of view of Infura, this is a huge liability. If Infura is compromised, then all of its users are immediately compromised.

This creates a dilemma between good performance and security for the users. This problem is not unique to Ethereum and appears in all proof-of-stake (PoS) systems. In this paper, we resolve this dilemma by constructing a PoS blockchain client which is both efficient and secure. We solve this problem by constructing a protocol that allows efficiently verifying the proof-of-stake blockchain without downloading the whole PoS. We call such constructions succinct *proofs of proof-of-stake*. These allow us to build *superlight clients*, clients whose communication complexity grows sublinearly with the lifetime of the system.

Contributions.

1. We give the first formal definition for succinct proof of proof-of-stake (PoPoS) protocols.
2. We put forth a solution to the long-standing problem of efficient PoS bootstrapping. Our solution is exponentially better than previous work.
3. We report on our implementation of a fully functional and highly performant superlight client for mainnet Ethereum. It is the first such construction for Ethereum. We measure and contrast the performance of our client against the currently proposed design for Ethereum.
4. We theoretically show our construction is secure for Ethereum and other PoS blockchains.

1.1 Construction Overview

Let us discuss the intuition about how our construction works. We start with the existing full node design and iteratively make it more lightweight.

Full nodes. A *full node* client boots holding only the genesis block and connects to other full nodes (known as *provers*) in order to synchronize to the latest tip. The full node downloads the entire chain block-by-block verifying each transaction in the process. This incurs high communication and computational complexity. Once the client verifies the latest tip, it has

[1] MetaMask has 21,000,000 monthly active users as of July 2022 [48] and is the most popular non-custodial wallet [16].

calculated the latest state and can answer user queries such as *"what is my balance?"*. In order for the full node to get to the correct tip, at least one connection to an honest peer is required (this is known as the *existential honesty* assumption [26, 25, 27, 44]).

Sync committees. Let's try to improve on the efficiency of the full node to make it a *light client*. PoS protocols typically proceed in *epochs* during which the validator set is fixed. In each epoch, a subset of validators is elected by the protocol as the epoch *committee*. This is a set of public keys. The security of the protocol assumes that the majority [17, 39, 19, 3] or super-majority [9, 13, 5, 51] of the committee members are honest during the epoch. The current committee signs the latest state. Therefore, the client does not need to download all the blocks, but instead only needs to download the latest state and verify the committee signatures on it. However, the stake changes hands in every epoch. Hence, to perform the verification at some later epoch, the client needs to keep track of the current committee. To help the client in this endeavor, the committee members of each epoch, while active, sign a *handover* message inaugurating the members of the new committee [28]. This enables the light client to discover the latest committee by processing a sequence of such handovers.

Optimistic light client construction. Light clients like these already exist. Regrettably, they still need to download all committees of the past to verify the current state. In this paper we propose better solutions. The first idea, which we call the *optimistic light client construction*, is for the prover to work as follows: For each committee, take its members, concatenate them all together, and hash them into one hash. The prover then sends this sequence of hashes (one for each committee) to the client. Since the client is connected to at least one honest prover, at least one of these provers will answer truthfully. If multiple provers give conflicting claims to the client, all it needs to do is to find which one is truthful. To do this, it compares the claims of all provers pairwise. If two provers disagree, the client focuses on the first point of disagreement in their hash sequences, and asks each prover to provide the respective handover signatures to substantiate their claim. Each prover reveals the committee attested by the hash at that point, the previous committee and the associated handover messages. Upon validating these messages, which can be done locally and efficiently, the client identifies the truthful party, and accepts its state. A lying prover will not be able to provide such a handover for an invalid committee. Once the client rejects the invalid committee claims, it will have calculated the latest committee, and can proceed from there as usual.

Superlight clients. Even though the complexity is concretely improved, the sequence of committee hashes still grows linearly with the lifetime of the protocol. To achieve sublinear complexity, we improve the procedure to find the first point of disagreement. To this end, our final PoPoS protocol requires each prover to organize its claimed sequence of committees – one per epoch – into a Merkle tree [43]. The roots of those trees are then sent over to the client, who compares them. Upon detecting disagreement at the roots, the client asks the provers to reveal the children of their respective roots. By repeating this process recursively on the mismatching children, it arrives at the *first point of disagreement* between the claimed committee sequences, in logarithmic number of steps. After the first point of disagreement is found, the client works similarly to the optimistic light client construction. This process achieves logarithmic communication.

Bridges. Our PoPoS construction has two main applications: *Superlight* proof-of-stake clients that can bootstrap very efficiently, and *trustless bridges* that allow the passing of information from one proof-of-stake chain to another. For bridging, we note that a trustless bridge is nothing more than an on-chain superlight client. It connects a source PoS blockchain to any other destination blockchain. To do this, a smart contract on the destination chain, which runs an implementation of our superlight client code (*e.g.*, in Solidity), is deployed. Whenever some information of interest appears on the source chain, any helpful but untrusted relayer can submit this information, together with the PoPoS proof to the smart contract. If the information is inaccurate, another relayer challenges the first one by participating in our interactive bisection game within a dispute period [40], submitting one transaction for every round of interaction of the PoPoS protocol. If both chains are PoS, the bridge can be made bidirectional by running PoPoS superlight clients on both chains. To incentivize on-chain participation, relayers who submit accurate information are rewarded, whereas relayers who submit inaccurate information need to put up a collateral which is slashed and is used to reward the challenger.

1.2 Implementation Overview

In addition to our theoretical contributions, we report on our open source implementation (spanning about 8,200 lines of TypeScript code and 2,300 downloads from the community) of our protocol for the Ethereum mainnet. Our implementation is fully functional and supports all RPC queries used by typical wallets, from simple payments to complex smart contract calls. It takes the form of a modular daemon that augments any existing Ethereum wallet's functionality (such as MetaMask's) to be trustless, without changing any user experience. We perform measurements using the light client protocol currently proposed for Ethereum. We find that this protocol, while much more efficient than a full node, is likely insufficient to support communication-, computation-, and battery-constrained devices such as browsers and mobile phones. Next, we measure the performance of our implementation of an *optimistic light client* for Ethereum that achieves significant gains over the traditional light client. We demonstrate this implementation is already feasible for resource-constrained devices. Lastly, our implementation includes a series of experiments introducing a *superlight client* for Ethereum that achieves exponential asymptotic gains over the optimistic light client. These gains constitute concrete improvements over the optimistic light client when the blockchain system is long-lived and has an execution history of a few years. We compare all three clients in terms of communication (bandwidth and latency), computation, and energy consumption.

1.3 Related Work

Table 1 compares the current paper with related work. Proof-of-work superlight clients have been explored in the interactive [35] and non-interactive [37] setting using various constructions [36, 32, 6]. Such constructions are backwards compatible [53, 38] and have been deployed in practice [18]. They have also been used in the context of bridges [33, 2, 40, 52]. Several proof-of-stake-specific clients [28, 14, 22] improve the efficiency of full nodes, but require linear communication. Chatzigiannis et al. [15] provide a survey of light clients.

Our construction is based on refereed proofs [12, 30, 31, 46]. PoPoS constructions can also be built via (recursive) SNARKs/STARKs [4, 24], some achieving constant communication. However, these are clients for chains that were designed from the start to be proof-friendly. Current attempts [50, 45] to retrofit popular PoS protocols (*e.g.*, Ethereum and Cosmos) with SNARK-based proofs are expensive (annual prover operating-cost of six to seven figures

Table 1 Comparison with previous works in terms of *asymptotic* $\widetilde{\Theta}$ communication complexity, interactivity, and cryptographic model. Interactivity is the number of rounds, ignoring constants. Low communication and rounds of interactivity are preferable. N: number of epochs. L: number of blocks per epoch. Common prefix parameter k is constant.

	SPV PoW [26]	**KLS** [35]	**FlyClient** [6]	**Superblocks** [37, 36]	**Full PoS** [39]	**Mithril** [14]	**Coda** [4]	**PoPoS** (this work)
Communication	NL	$\log(NL)$	poly $\log(NL)$	$\log(NL)$	NL	$N+L$	1	$\log(N)$
Interactivity	1	$\log(NL)$	1	1	1	1	1	$\log(N)$
Work/stake	work	work	work	work	stake	stake	both	stake
Model	RO	RO	RO	RO	standard	RO	CRS	standard
Primitives	hash	hash	hash	hash	hash, sig	hash, sig, ZK	hash, sig, ZK	hash, sig

USD).[2] Additionally, since [50, 45] do not include recursive proving/verification, they require the validator to remain online (to avoid linear overhead), which may be a suitable assumption for bridges, but not for bootstrapping light clients (*e.g.*, intermittently online mobile wallets). Unlike proof-based approaches, our protocol does not require changes in the PoS protocol, and relies on simple primitives such as hashes and signatures. Our prover is stateless and adds only a few milliseconds of computation time to an existing full node. Our light client allows for bootstrapping from genesis with sublinear overhead.

Some clients obtain a *checkpoint* [41, 8] on a recent block from a trusted source, after which they download only a constant number of block headers to identify the tip. Our construction allows augmenting these clients so that they can *succinctly verify* the veracity of a checkpoint without relying on a trusted third party.

1.4 Outline

We present our theoretical protocol in a *generic* PoS framework, which typical proof-of-stake systems fit into. We prove our protocol is secure if the underlying blockchain protocol satisfies certain simple and straightforward *axioms*. Many popular PoS blockchains can be made to fit within our axiomatic framework. We define our desired primitive, the proof of proof-of-stake (PoPoS), together with the axioms required from the underlying PoS protocol in Sec. 3. We iteratively build and present our construction in Secs. 4 and 5. We present the security claims in Sec. 8. For concreteness, and because it is the most prominent PoS protocol, we give a concrete construction of our protocol for Ethereum in Sec. 6. Ethereum is the most widely adopted PoS protocol. Interestingly, Ethereum directly satisfies our axiomatic framework and does not require any changes on the consensus layer at all. The applicability of our framework to other PoS chains such as Ouroboros (Cardano), Algorand, and Snow White are discussed in the full version of this paper [1].

The description of our implementation and the relevant experimental measurements showcasing the advantages of our implementation are presented in Sec. 7.

[2] For example, according to [50, Section 6], proving consensus (of 128 validators) on *one* Cosmos block takes 18 seconds on 32 instances of Amazon AWS `c5.24xlarge`. (We are unable to independently reproduce this finding because the authors of zkBridge have not disclosed their code.) At Cosmos' block rate of 1 block per second, that would require 18×32 continuously operating `c5.24xlarge` instances, costing annually $12,967,488 on Amazon AWS (annual pricing, `us-east-1` region, June 2023), or $1,749,600 on Hetzner's equivalents. Scaling this proportionally for Ethereum ($\times 4$ for sync committee size 512, /12 for block rate 1 block per 12 seconds) yields an estimate of $583,333 annually.

2 Preliminaries

Proof-of-stake. Our protocols work in the proof-of-stake (PoS) setting. In a PoS protocol, participants transfer value and maintain a balance sheet of *stake*, or *who owns what*, among each other. It is assumed that the *majority of stake* is honestly controlled at every point in time. The PoS protocol uses the current stake *distribution* to establish consensus. The exact mechanism by which consensus is reached varies by PoS protocol. Our PoPoS protocol works for popular PoS flavours.

Primitives. Participants in our PoS protocol transfer stake by *signing* transactions using a signature scheme [34]. The public key associated with each validator is known by everyone. The signatures are key-evolving, and honest validators delete their old keys after signing [29, 19].[3] We also use a collision resistant hash function. We highlight that it does not need to be treated in the Random Oracle model, and no trusted setup is required for our protocol (beyond what the underlying PoS protocol may need).

Types of nodes. The stakeholders who participate in maintaining the system's consensus are known as *validators*. In addition to those, other parties, who do not participate in maintaining consensus, can join the system, download its full history, and discover its current state. These are known as *full nodes*. Clients that are interested in joining the system and learning a small part of the system state (such as their user's balance) without downloading everything are known as *light clients*. Both full nodes and light clients can join the system at a later time, after it has already been executing for some duration $|\mathcal{C}|$. A late-joining light client or full node must *bootstrap* by downloading some data from its peers. The amount of data the light client downloads to complete the bootstrapping process is known as its *communication complexity*. A light client is *succinct* if its communication complexity is $\mathcal{O}(\operatorname{poly}\log(|\mathcal{C}|))$ in the lifetime $|\mathcal{C}|$ of the system. Succinct light clients are also called *superlight clients*. The goal of this paper is to develop a PoS superlight client.

Time. The protocol execution proceeds in discrete *epochs*, roughly corresponding to moderate time intervals such as one day. Epochs are further subdivided into *rounds*, which correspond to shorter time durations during which a message sent by one honest party is received by all others. In our analysis, we assume synchronous communication. The validator set stays fixed during an epoch, and it is known one epoch in advance. The validator set of an epoch is determined by the snapshot of stake distribution at the beginning of the previous epoch. To guarantee an honest majority of validators at any epoch, we assume a *delayed honest majority* for a duration of *two epochs*: Specifically, if a snapshot of the current stake distribution is taken at the beginning of an epoch, this snapshot satisfies the honest majority assumption for a duration of two full epochs. Additionally, we assume that the adversary is *slowly adaptive*: She can corrupt any honest party, while respecting the honest majority assumption, but that corruption only takes place two epochs later. This assumption will be critical in our construction of *handover* messages that allow members of one epoch to inaugurate a committee representing the next epoch (*cf.* Sec. 4).

[3] Instead of key-evolving signatures, Ethereum relies on a concept called *weak subjectivity* [8]. This alternative assumption can also be used in the place of key-evolving signatures to prevent posterior corruption attacks [20].

The prover/verifier model. The bootstrapping process begins with a light client connecting to its full node peers to begin synchronizing. During the synchronization process, the full nodes are trying to convince the light client of the system's state. In this context, the light client is known as the *verifier* and the full nodes are known as the *provers*. As usual, we assume the verifier is connected to at least one honest prover. The verifier queries the provers about the state of the system, and can exchange multiple messages to interrogate them about the truth of their claims during an *interactive protocol*.

Assumptions. We make two central assumptions: Firstly, that the light client can communicate *interactively* with full nodes. This is contrary to, for example, proof-based clients. Interactivity incurs a penalty when our light client runs on-chain, because it requires receiving data over the course of multiple transactions. This is expensive in gas and time consuming in delays. Secondly, that the light client has *at least one honest* connection. Many protocols assume this. For example, a Bitcoin full node is not secure if all connections are dishonest. In current systems, such as Ethereum, light clients typically connect to RPC nodes instead of full nodes. It is better to trust at least *one among many* RPC connections is honest (as opposed to having a single RPC connection), but one may still become eclipsed. To resolve this concern, we advocate for light clients to connect to full nodes directly, which would make this assumption more reasonable. Due to these two assumptions, we caution the reader to be aware of the current limitations of our work, understanding it is not always applicable.

Notation. We use ϵ and $[\,]$ to mean the empty string and empty sequence. By $x \,\|\, y$, we mean the string concatenation of x and y encoded in a way that x and y can be unambiguously retrieved. We denote by $|\mathcal{C}|$ the length of the sequence $\mathcal{C}$; by $\mathcal{C}[i]$ the i^{th} (zero-based) element of the sequence, and by $\mathcal{C}[-i]$ the i^{th} element from the end. We use $\mathcal{C}[i{:}j]$ to mean the subarray of $\mathcal{C}$ from the i^{th} element (inclusive) to the j^{th} element (exclusive). Omitting i takes the sequence to the beginning, and omitting j takes the sequence to the end. We write $A \preccurlyeq B$ to mean that the sequence A is a prefix of B. We use λ to denote the security parameter. Following Go notation, in our multi-party algorithms, we use $m \dashrightarrow A$ to indicate that message m is sent to party A and $m \dashleftarrow A$ to indicate that message m is received from party A.

Ledgers. The consensus protocol attempts to maintain a unified view of a *ledger* $\mathbb{L}$. The ledger is a sequence of *transactions* $\mathbb{L} = (\mathsf{tx}_1, \mathsf{tx}_2, \ldots)$. Each validator and full node has a different view of the ledger. We denote the ledger of party P at round r as $\mathbb{L}^P_r$. Nodes joining the protocol, whether they are validators, full nodes, or (super)light clients, can also *write* to the ledger by asking for a transaction to be included. In a secure consensus protocol, all honestly adopted ledgers are prefixes of one another. We denote the longest among these ledgers as $\mathbb{L}^{\cup}_r$, and the shortest among them as $\mathbb{L}^{\cap}_r$. We will build our protocol on top of PoS protocols that are secure. A *secure* consensus protocol enjoys the following two virtues:

▶ **Definition 1** (Consensus Security). *A consensus protocol is* secure *if it is:*

1. ***Safe:*** *For any honest parties P_1, P_2 and rounds $r_1 \le r_2$: $\mathbb{L}^{P_1}_{r_1} \preccurlyeq \mathbb{L}^{P_2}_{r_2}$.*
2. ***Live:*** *If all honest validators attempt to* write *a transaction during u consecutive rounds $r_1, \ldots, r_u$, it is included in $\mathbb{L}^P_{r_u}$ of any honest party P.*

Transactions. A transaction encodes an update to the system's state. For example, a transaction could indicate a value transfer of 5 units from Alice to Bob. Different systems use different transaction formats, but the particular format is unimportant for our purposes.

A transaction can be applied on the current *state* of the system to reach a new state. Given a state st and a transaction tx, the new state is computed by applying the state transition function δ to the state and transaction. The new state is then $\mathsf{st}' = \delta(\mathsf{st}, \mathsf{tx})$. For example, in Ethereum, the state of the system encodes a list of balances of all participants [7, 49]. The system begins its lifetime by starting at a genesis state st_0. A ledger also corresponds to a particular system state, the state obtained by applying its transactions iteratively to the genesis state. Consider a ledger $\mathbb{L} = (\mathsf{tx}_1 \cdots \mathsf{tx}_n)$. Then the state of the system is $\delta(\cdots\delta(\mathsf{st}_0, \mathsf{tx}_1), \cdots, \mathsf{tx}_n)$. We use the shorthand notation δ^* to apply a sequence of transactions $\overline{\mathsf{tx}} = \mathsf{tx}_1 \cdots \mathsf{tx}_n$ to a state. Namely, $\delta^*(\mathsf{st}_0, \overline{\mathsf{tx}}) = \delta(\cdots\delta(\mathsf{st}_0, \mathsf{tx}_1), \cdots, \mathsf{tx}_n)$.

Because the state of the system is large, the state is compressed using an authenticated data structure (*e.g.*, Merkle Tree [43]). We denote by $\langle \mathsf{st} \rangle$ the state *commitment*, which is this short representation of the state st (*e.g.*, Merkle Tree root). Given a state commitment $\langle \mathsf{st} \rangle$ and a transaction tx, it is possible to calculate the state commitment $\langle \mathsf{st}' \rangle$ to the new state $\mathsf{st}' = \delta(\mathsf{st}, \mathsf{tx})$. However, this calculation may require a small amount of auxiliary data π such as a Merkle tree proof of inclusion of certain elements in the state commitment $\langle \mathsf{st} \rangle$. We denote the transition that is performed at the state commitment level by the *succinct transition function* $\langle \delta \rangle$. Concretely, we will write that $\langle \delta(\mathsf{st}, \mathsf{tx}) \rangle = \langle \delta \rangle (\langle \mathsf{st} \rangle, \mathsf{tx}, \pi)$. This means that, if we take state st and apply transaction tx to it using the transition function δ, and subsequently calculate its commitment using the $\langle \cdot \rangle$ operator, the resulting state commitment is the same as the one obtained by applying the succinct transition function $\langle \delta \rangle$ to the state commitment $\langle \mathsf{st} \rangle$ and transaction tx using the auxiliary data π. If the auxiliary data is incorrect, the function $\langle \delta \rangle$ returns $\perp$ to indicate failure. If the state commitment uses a secure authenticated data structure such as a Merkle tree, we can only find a unique π that makes the $\langle \delta \rangle$ function run successfully.

3 The PoPoS Primitive

The PoPoS abstraction. Every verifier $\mathcal{V}$ online at some round r holds a state commitment $\langle \mathsf{st} \rangle_r^{\mathcal{V}}$. To learn about this recent state, the verifier connects to provers $\mathcal{P} = \{P_1, P_2, \cdots, P_q\}$. All provers except one honest party can be controlled by the adversary, and the verifier does not know which party among the provers is honest (the verifier is assumed to be honest). The honest provers are always online. Each of them maintains a ledger $\mathbb{L}_i$. They are consistent by the safety of the underlying PoS protocol. Upon receiving a query from the verifier, each honest prover sends back a state commitment corresponding to its current ledger. However, the adversarial provers might provide incorrect or outdated commitments that are different from those served by their honest peers. To identify the correct commitment, the light client mediates an *interactive* protocol among the provers:

▶ **Definition 2** (Proof of Proof-of-Stake). *A* Proof of Proof-of-Stake protocol *(PoPoS) for a PoS consensus protocol is a pair of interactive probabilistic polynomial-time algorithms (P, V). The algorithm P is the* honest prover *and the algorithm V is the* honest verifier. *The algorithm P is ran by an online full node, while V is a light client booting up for the first time holding only the genesis state commitment $\langle \mathsf{st}_0 \rangle$. The protocol is executed between V and a set of provers $\mathcal{P}$. After completing the interaction, V returns a state commitment $\langle \mathsf{st} \rangle$.*

Security of the PoPoS protocol. The goal of the verifier is to output a state commitment consistent with the view of the honest provers. This is reflected by the following security definition of the PoPoS protocol.

▶ **Definition 3** (State Security). *Consider a PoPoS protocol* (P, V) *executed at round* r*, where* V *returns* $\langle \mathsf{st} \rangle$*. It is* secure *with* parameter ν *if there exists a ledger* $\mathbb{L}$ *such that* $\langle \mathsf{st} \rangle = \delta^*(\mathsf{st}_0, \mathbb{L})$*, and* $\mathbb{L}$ *satisfies:*

- ***Safety:*** *For all rounds* $r' \geq r + \nu$*:* $\mathbb{L} \preccurlyeq \mathbb{L}^{\cup}_{r'}$*.*
- ***Liveness:*** *For all rounds* $r' \leq r - \nu$*:* $\mathbb{L}^{\cap}_{r'} \preccurlyeq \mathbb{L}$*.*

State security implies that the commitment returned by a verifier corresponds to a state recently obtained by the honest provers.

4 The Optimistic Light Client

Before we present our succinct PoPoS protocol, we introduce sync committees and handover messages, two necessary components we use in our construction. We also propose a highly performant optimistic light client as a building block for the superlight clients.

Sync committees. To allow the verifier to achieve state security, we introduce a *sync committee* (first proposed in the context of PoS sidechains [28]). Each committee is elected for the duration of an epoch, and contains a subset, of fixed size m, of the public keys of the validators associated with that epoch. The committee of the next epoch is determined in advance at the beginning of the previous epoch. All honest validators agree on this committee. The validators in the sync committee are sampled from the validator set of the corresponding epoch in such a manner that the committee retains honest majority during the epoch. The exact means of sampling are dependent on the PoS implementation. One way to construct the sync committee is to sample uniformly at random from the underlying stake distribution using the epoch randomness of the PoS protocol [39, 23]. The first committee S^0 is recorded by the genesis state st_0. We denote the set of public keys of the sync committee assigned to epoch $j \in \mathbb{N}$ by S^j, and each committee member public key within S^j by S^j_i, $i \in \mathbb{N}$.

Handover signatures. During each epoch j, each honest committee member S^j_i of epoch j signs the tuple $(j+1, S^{j+1})$, where $j+1$ is the next epoch index and S^{j+1} is the set of all committee member public keys of epoch $j+1$. We let σ^j_i denote the signature of S^j_i on the tuple $(j+1, S^{j+1})$. This signature means that member S^j_i approves the inauguration of the next epoch committee. We call those *handover signatures*[4], as they signify that the previous epoch committee *hands over* control to the next committee. When epoch $j+1$ starts, the members of the committee S^j assigned to epoch j can no longer use their keys to create handover signatures.[5] As soon as more than $\frac{m}{2}$ members of S^j have approved the inauguration of the next epoch committee, the inauguration is ratified. This collection of signatures for the handover between epoch j and $j+1$ is denoted by Σ^{j+1}, and is called the *handover proof.* A *succession* $\mathbb{S} = (\Sigma^1, \Sigma^2, \ldots, \Sigma^j)$ at an epoch j is the sequence of all handover proofs across an execution until the beginning of the epoch.

In addition to the handover signature, at the beginning of each epoch, honest committee members sign the *state commitment* corresponding to their ledger. When the verifier learns the latest committee, these signatures enable him to find the current state commitment.

[4] Handover signatures between PoS epochs were introduced in the context of PoS sidechains [28]. Some practical blockchain systems already implement similar handover signatures [54, 42].

[5] This assumption can be satisfied using key-evolving signatures [29, 19], social consensus [8], or a static honest majority assumption.

A naive linear client. Consider a PoPoS protocol, where each honest prover gives the verifier a state commitment and signatures on the commitment from the latest sync committee S^{N-1}, where N is the number of epochs (and $N-1$ is the last epoch). To convince the verifier that S^{N-1} is the correct latest committee, each prover also shares the sync committees $S^0 \dots S^{N-2}$ and the associated handover proofs in its view. The verifier knows S_0 from the genesis state st_0, and can verify the committee members of the future epochs iteratively through the handover proofs. Namely, upon obtaining the sync committee S^j, the verifier accepts a committee S^{j+1} as the correct committee assigned to epoch $j+1$, if there are signatures on the tuple $(j+1, S^{j+1})$ from over half of the committee members in S^j. Repeating the process above, the verifier can identify the correct committee for the last epoch. After identifying the latest sync committee, the verifier checks if the state commitment provided by a prover is signed by over half of the committee members. If so, he accepts the commitment.

It is straightforward to show that this strawman PoPoS protocol (which we abbreviate as TLC) is secure (Def. 3) under the following assumptions:

1. The underlying PoS protocol satisfies safety and liveness.
2. The majority of the sync committee members are honest.

When all provers are adversarial, the verifier might not receive any state commitment from them. Even though generally at least one prover is assumed to be honest, the strawman protocol does not need this for the *correctness* of the commitment accepted by the verifier, since the verifier validates each sync committee assigned to consecutive epochs, and does not accept commitments not signed by over $\frac{m}{2}$ members of the latest committee.

Regrettably, the strawman protocol is $\mathcal{O}(|\mathcal{C}|)$ and not succinct: To identify the lastest sync committee, the verifier has to download each sync committee since the genesis block. In the rest of this paper, we will improve this protocol to make it succinct.

The optimistic light client (OLC). We now reduce the communication complexity of the verifier. In this version of the protocol, instead of sharing the sync committees $S^0 \dots S^{N-2}$ and the associated handover proofs, each honest prover P sends a sequence of *hashes* $h^1 \dots h^{N-1}$ corresponding to the sync committees $S^0 \dots S^{N-1}$. Subsequently, to prove the correctness of the state commitment, the prover P reveals the latest sync committee S^{N-1} assigned to epoch $N-1$ and the signatures by its members on the commitment. Upon receiving the committee S^{N-1}, the verifier checks if the hash of S^{N-1} matches h^{N-1}, and validates the signatures on the commitment.

Unfortunately, an adversarial prover P^* can claim an incorrect committee $S^{*,N-1}$, whose hash $h^{*,N-1}$ disagrees with h^{N-1} returned by P. This implies a disagreement between the two hash sequences received from P and P^*. The verifier can exploit this discrepancy to identify the truthful party that returned the correct committee. Towards this goal, the verifier iterates over the two hash sequences, and finds the *first point of disagreement.* Let j be the index of this point such that $h^j \neq h^{*,j}$ and $h^i = h^{*,i}$ for all $i < j$. The verifier then requests P to reveal the committees S^j and S^{j-1} at the preimage of h^j and h^{j-1}, and to supply a handover proof Σ^j for S^{j-1} and S^j. He also requests P^* to reveal the committees $S^{*,j}$ and $S^{*,j-1}$ at the preimage of $h^{*,j}$ and $h^{*,j-1}$, and to supply a handover proof $\Sigma^{*,j}$ for $S^{*,j-1}$ to $S^{*,j}$. As $h^{j-1} = h^{*,j-1}$ by definition, the verifier is convinced that the committees S^{j-1} and $S^{*,j-1}$ revealed by P and P^* are the same.

Finally, the verifier checks whether the committees $S^{*,j}$ and S^j were inaugurated by the previous committee S^{j-1} using the respective handover proofs Σ^j and $\Sigma^{*,j}$. Since S^{j-1} contains over $\frac{m}{2}$ honest members that signed only the correct committee S^j assigned to epoch j, adversarial prover P^* cannot create a handover proof with sufficiently many signatures

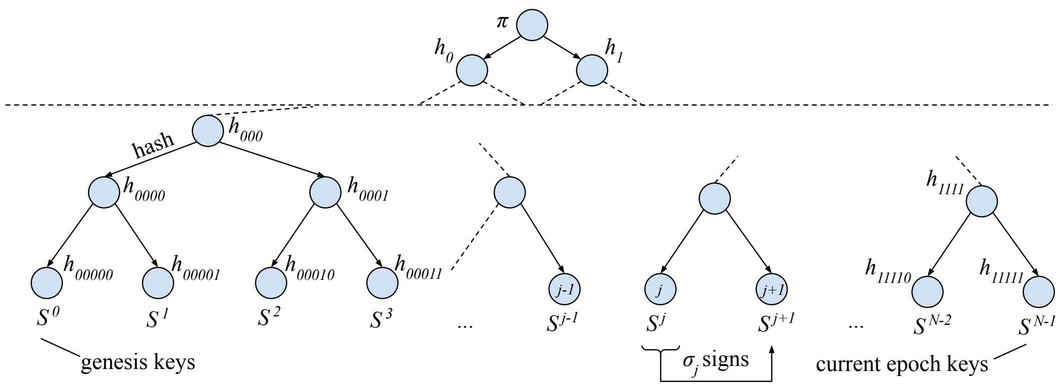


Figure 1 The *handover tree*, the central construction of our protocol. The root of the Merkle tree is the initial proof π. During the bisection game, the signatures between the challenge node j and its neighbours $j-1$ and $j+1$ are validated.

inaugurating $S^{*,j}$. Hence, the handover from S^{j-1} to $S^{*,j}$ will not be ratified $\Sigma^{*,j}$, whereas the handover from S^{j-1} to S^j will be ratified by Σ^j. Consequently, the verifier will identify P as the truthful party and accept its commitment.

In the protocol above, security of the commitment obtained by the prover relies crucially on the existance of an honest prover. Indeed, when all provers are adversarial, they can collectively return the *same* incorrect state commitment and the *same* incorrect sync committee for the latest epoch. They can then provide over $\frac{m}{2}$ signatures by this committee on the incorrect commitment. In the absence of an honest prover to challenge the adversarial ones, the verifier would believe in the validity of an incorrect commitment.

The optimistic light client reduces the communication load of sending over the whole sync committee sequence by representing each committee with a constant size hash. However, it is still $\mathcal{O}(|\mathcal{C}|)$ as the verifier has to do a linear search on the hashes returned by the two provers to identify the first point of disagreement. To support a truly succinct verifier, we will next work towards an interactive PoPoS protocol based on bisection games.

5 The Superlight Client

Trees and mountain ranges. Before describing the succinct PoPoS protocol and the superlight client, we introduce the data structures used by the bisection games.

Suppose the number of epochs N is a power of two. The honest provers organize the committee sequences for the past epochs into a Merkle tree called the *handover tree* (Fig. 1). The j^{th} leaf of the handover tree contains the committee S^j of the j^{th} epoch. A handover tree consisting of leaves $S^0, \ldots, S^{N-1}$ is said to be *well-formed* with respect to a succession $\mathbb{S}$ if it satisfies the following properties:

1. The leaves are syntactically valid. Every j^{th} leaf contains a sync committee S^j that consists of m public keys.
2. The first leaf corresponds to the known *genesis* sync committee S^0.
3. For each $j = 1 \ldots N-1$, Σ^j consists of over $\frac{m}{2}$ signatures by members of S^{j-1} on (j, S^j).

Every honest prover holds a succession of handover signatures attesting to the inauguration of each sync committee in its handover tree after S^0. These successions might be different for every honest prover as any set of signatures larger than $\frac{m}{2}$ by S^j can inaugurate S^{j+1}. However, the trees are the same for all honest parties, and they are well-formed with respect to the succession held by each honest prover.

When the number N of epochs is not a power of two, provers arrange the past sync committees into Merkle mountain ranges (MMRs) [47, 21]. An MMR is a list of Merkle trees, whose sizes are decreasing powers of two. To build an MMR, a prover first obtains a binary representation $2^{q_1} + \ldots + 2^{q_n}$ of N, where $q_1 > \ldots > q_n$. It then divides the sequence of sync committees into n subsequences, one for each q_i. For $i \geq 1$, the i^{th} subsequence contains the committees $S^{\sum_{n=1}^{i-1} 2^{q_i}}, \ldots, S^{(\sum_{n=1}^{i} 2^{q_i})-1}$. Each i^{th} subsequence is organized into a distinct Merkle tree $\mathcal{T}_i$, whose root, denoted by $\langle \mathcal{T}_i \rangle$, is called a *peak*. These peaks are all hashed together to obtain the root of the MMR. We hereafter refer to the index of each leaf in these Merkle trees with the epoch of the sync committee contained at the leaf. (For instance, if there are two trees with sizes 4 and 2, the leaf indices in the first tree are $0, 1, 2, 3$ and the leaf indices in the second tree are 4 and 5.) The MMR is said to be *well formed* if each constituent tree is well-formed (but, of course, only the first leaf of the first tree needs to contain the genesis committee). To ensure succinctness, only the peaks and a small number of leaves, with their respective inclusion proofs, will be presented to the verifier during the following bisection game.

Different state commitments. We begin our construction of the full PoPoS protocol (abbreviated SLC for *Super Light Client*) by describing the first messages exchanged between the provers $\mathcal{P}$ and the verifier. Each honest prover first shares the state commitment signed by the latest sync committee at the beginning of the last epoch $N-1$. If all commitments received by the verifier are the same, by existential honesty, the verifier rests assured this commitment is correct, *i.e.*, it corresponds to the ledger of the honest provers at the beginning of the epoch. Otherwise, the verifier requests from each prover in $\mathcal{P}$: (i) the MMR peaks $\langle \mathcal{T} \rangle_i$, $i \in [n]$ held by the prover, where n is the number of peaks, (ii) the latest sync committee S^{N-1}, (iii) a Merkle inclusion proof for S^{N-1} with respect to the last peak $\langle \mathcal{T} \rangle_n$, and (iv) signatures by the committee members in S^{N-1} on the state commitment given by the prover.

Upon receiving these messages, the verifier first checks if there are more than $\frac{m}{2}$ valid signatures by the committee members in S^{N-1} on the state commitment. It then verifies the Merkle proof for S^{N-1} with respect to $\langle \mathcal{T} \rangle_n$. As the majority of the committee members in S^{N-1} are honest, it is not possible for different state commitments to be signed by over half of S^{N-1}. Hence, if the checks above succeed for two provers P and P^* that returned different commitments, one of them (P^*) must be an adversarial prover, and must have claimed an incorrect sync committee $S^{*,N-1}$ for the last epoch. Moreover, as the Merkle proofs for both $S^{*,N-1}$ and S^{N-1} verify against the respective peaks $\langle \mathcal{T} \rangle_n$ and $\langle \mathcal{T} \rangle_n^*$, these peaks must be different. Since the two provers disagree on the roots and there is only one well-formed MMR at any given epoch, therefore one of the provers does not hold a well-formed MMR. This reduces the problem of identifying the correct state commitment to detecting the prover that has a well-formed MMR behind its peaks.

Bisection game. To identify the honest prover with the well-formed MMR, the verifier (Alg. 1) initiates a bisection game between P and P^* (Alg. 2). Suppose the number of epochs N is a power of two. Each of the two provers claims to hold a tree with size N (otherwise, since the verifier knows N by his local clock, the prover with a different size Merkle tree loses the game.) During the game, the verifier aims to locate the first point of disagreement between the alleged sync committee sequences at the leaves of the provers' Merkle trees, akin to the improved optimistic light client (Sec. 4).

The game proceeds in a binary search fashion similar to refereed delegation of computation [12, 11, 30]. Starting at the Merkle roots $\langle \mathcal{T} \rangle$ and $\langle \mathcal{T} \rangle^*$ of the two trees, the verifier traverses an identical path on both trees until reaching a leaf with the same index. This

Algorithm 1 Run by the verifier during the bisection game to identify the first point of disagreement between the provers' leaves (*cf.* Fig. 2). Here, P and P^* denote the honest and adversarial provers, whereas $\langle\mathcal{T}\rangle$ and $\langle\mathcal{T}\rangle^*$ denote the roots of their respective Merkle trees with size ℓ.

```
1:  function FindDisagreement($P, \langle\mathcal{T}\rangle, P^*, \langle\mathcal{T}\rangle^*, \ell$)
2:      $h_c, h_c^* \leftarrow \langle\mathcal{T}\rangle, \langle\mathcal{T}\rangle^*$
3:      while $\ell > 1$ do
4:          $(h_0, h_1) \dashleftarrow P$; $(h_0^*, h_1^*) \dashleftarrow P^*$
5:          if $h_c \neq H(h_0 \,\|\, h_1)$ then return                    ▷ $P$ loses
6:          if $h_c^* \neq H(h_0^* \,\|\, h_1^*)$ then return                 ▷ $P^*$ loses
7:          if $h_0 \neq h_0^*$ then
8:              $h_c^*, h_c \leftarrow h_0^*, h_0$; $(\text{open}, 0) \dashrightarrow P$; $(\text{open}, 0) \dashrightarrow P^*$
9:          else
10:             $h_c^*, h_c \leftarrow h_1^*, h_1$; $(\text{open}, 1) \dashrightarrow P$; $(\text{open}, 1) \dashrightarrow P^*$
11:         $\ell \leftarrow \ell // 2$
12:     $S \dashleftarrow P$; $S^* \dashleftarrow P^*$
13:     return $S, S^*$
```

Algorithm 2 Run by an honest prover during the bisection game to reply to the verifier V's queries. The sequence $S^0, \ldots, S^{N-1}$ denotes the sync committees in the prover's view.

```
1:  function ReplyToVerifier($S^0, \ldots, S^{N-1}$)
2:      $\mathcal{T} \leftarrow \text{MakeMT}(S^0, \ldots, S^{N-1})$; $\mathcal{T}.\text{root} \dashrightarrow V$; $j \leftarrow 0$
3:      while $\mathcal{T}.\text{size} > 1$ do
4:          $(\mathcal{T}.\text{left.root}, \mathcal{T}.\text{right.root}) \dashrightarrow V$; $(\text{open}, i) \dashleftarrow V$
5:          if $i = 0$ then
6:              $\mathcal{T} \leftarrow \mathcal{T}.\text{left}$
7:          else
8:              $\mathcal{T} \leftarrow \mathcal{T}.\text{right}$
9:          $j \leftarrow 2j + i$
10:     $S^j \dashrightarrow V$
```

leaf corresponds to the first point of disagreement. At each step of the game, the verifier asks the provers to reveal the children of the *current* node, denoted by h_c and h_c^* on the respective trees (Alg. 2, l. 4). Initially, $h_c = \langle\mathcal{T}\rangle$ and $h_c^* = \langle\mathcal{T}\rangle^*$ (Alg. 1, l. 2). Upon receiving the alleged left and right child nodes h_0^* and h_1^* from P^*, and h_0, h_1 from P, he checks if $h_c = H(h_0 \,\|\, h_1)$ and $h_c^* = H(h_0^* \,\|\, h_1^*)$, where H is the collision-resistant hash function used to construct the Merkle trees (Alg. 1, ll. 5 and 6). The verifier then compares h_0 with h_0^*, and h_1 with h_1^* to determine if the disagreement is on the left or the right child (Alg. 1, ll. 7 and 9). Finally, he descends into the first disagreeing child, and communicates this decision to the provers (Alg. 2, l. 4); so that they can update the current node that will be queried in the next step of the bisection game (Alg. 2, ll. 6 and 8).

Upon reaching a leaf at some index j, the verifier asks both provers to reveal the alleged committees S^j and $S^{*,j}$ at the pre-image of the respective leaves. If $j = 1$, he inspects whether S^j or $S^{*,j}$ matches S^0. The prover whose alleged first committee is not equal to S^0 loses the game.

If $j > 1$, the verifier also requests from the provers (i) the committees at the $(j-1)^{\text{th}}$ leaves, (ii) their Merkle proofs with respect to $\langle\mathcal{T}\rangle$ and $\langle\mathcal{T}\rangle^*$, and (iii) the handover proofs Σ^j and $\Sigma^{*,j}$. The honest prover responds with (i) S^{j-1} assigned to epoch $j-1$, (ii) its Merkle proof with respect to $\langle\mathcal{T}\rangle$, and (iii) its own view of the handover proof Σ^j (which might be different from other provers). Upon checking the Merkle proofs, the verifier is now

Algorithm 3 Run by the verifier to identify the first different peak in the MMRs of the two provers. Here, $\langle\mathcal{T}\rangle_{1,\ldots,n}$ and $\langle\mathcal{T}\rangle^*_{1,\ldots,n}$ denote the peaks of the honest and adversarial provers respectively.

```
1: function BisectionGame($P, P^*$)
2:     $\langle\mathcal{T}\rangle_{1,\ldots,n} \dashleftarrow P$
3:     $\langle\mathcal{T}\rangle^*_{1,\ldots,n} \dashleftarrow P^*$
4:     for $i = 1$ to $n$ do
5:         if $\langle\mathcal{T}\rangle_i \neq \langle\mathcal{T}\rangle^*_i$ then
6:             $\ell \leftarrow$ size of the $i^{\text{th}}$ Merkle Tree
7:             return FindDisagreement($P, \langle\mathcal{T}\rangle_i, P^*, \langle\mathcal{T}\rangle^*_i, \ell$)
```

convinced that the committees S^{j-1} and $S^{*,j-1}$ revealed by P and P^* are the same, since their hashes match. The verifier subsequently checks if Σ^j contains more than $\frac{m}{2}$ signatures by the committee members in S^{j-1} on (j, S^j), and similarly for P^*.

The prover that fails any of checks by the verifier loses the bisection game. If one prover loses the game, and the other one does not fail any checks, the standing prover is designated the winner. If neither prover fails any of the checks, then the verifier concludes that there are over $\frac{m}{2}$ committee members in S^{j-1} that signed different future sync committees (*i.e.*, signed both (j, S^j) and $(j, S^{*,j})$, where $(j, S^j) \neq (j, S^{*,j})$). This implies S^{j-1} is not the correct sync committee assigned to epoch $j-1$, and both provers are adversarial. In this case, both provers lose the bisection game. In any case, at most one prover can win the bisection game.

Bisection games on Merkle mountain ranges. When the number of epochs N is not a power of two, the verifier first obtains the binary decomposition $\sum_{i=1}^{n} 2^{q_i} = N$, where $q_1 > \ldots > q_n$. Then, for each prover P, he checks if there are n peaks returned. For two provers P and P^* that have n peaks but returned different commitments, the verifier compares the peaks $\langle\mathcal{T}\rangle_i$ of P with $\langle\mathcal{T}\rangle^*_i$ of P^*, and identifies the first different peak (Alg. 3). It then plays the bisection game as described above on the identified Merkle trees. The only difference with the game above is that if the disagreement is on the first leaf j of a later tree, then the Merkle proof for the previous leaf $j-1$ is shown with respect to the peak of the previous tree.

Prover complexity. Given all past sync committees, the prover constructs the MMR in linear time. The MMR is updated in an online fashion as time evolves. Every time a new sync committee appears, it is appended to the tree in $\log N$ time. The space required to store the MMR is linear.

Tournament. When there are multiple provers, the verifier interacts with them sequentially in pairs, in a tournament fashion. It begins by choosing two provers P_1 and P_2 with different state commitments from the set $\mathcal{P}$ (Alg. 4, l. 7). The verifier then *pits one against the other*, by facilitating a bisection game between P_1 and P_2, and decides which of the two provers loses (Alg. 4, l. 8). (There can be at most one winner at any bisection game). He then eliminates the loser from the tournament, and chooses a new prover with a different state commitment than the winner's commitment from the set $\mathcal{P}$ to compete against the winner. In the event that both provers lose, the verifier eliminates both provers, and continues the tournament with the remaining ones by sampling two new provers with different state commitments. This process continues until all provers left have the same state commitment. This commitment is adopted as the correct one. A tournament started with q provers terminates after $O(q)$

Algorithm 4 Tournament conducted by the verifier among provers $\mathcal{P}$ to identify the state commitment $\langle \mathsf{st} \rangle$. The verifier uses BisectionGame (*cf.* Fig. 2, Algs. 1 and 3) between two provers and deduce at most *one* winner. Here, pop removes and returns an arbitrary element of a set.

```
1:  function Tournament($\mathcal{P}$)
2:      $P \leftarrow \mathrm{pop}(\mathcal{P})$; $\mathrm{good} \leftarrow \{P\}$; $\langle \mathsf{st} \rangle \leftarrow P.\langle \mathsf{st} \rangle$
3:      for $P \in \mathcal{P}$ do
4:          if $\langle \mathsf{st} \rangle = P.\langle \mathsf{st} \rangle$ then
5:              $\mathrm{good} \leftarrow \mathrm{good} \cup \{P\}$
6:              continue
7:          for $P^* \in \mathrm{good}$ do
8:              if BisectionGame$(P, P^*) = P$ then
9:                  $\mathrm{good} \leftarrow \{P\}$; $\langle \mathsf{st} \rangle \leftarrow P.\langle \mathsf{st} \rangle$
10:                 break
11:     return $\langle \mathsf{st} \rangle$
```

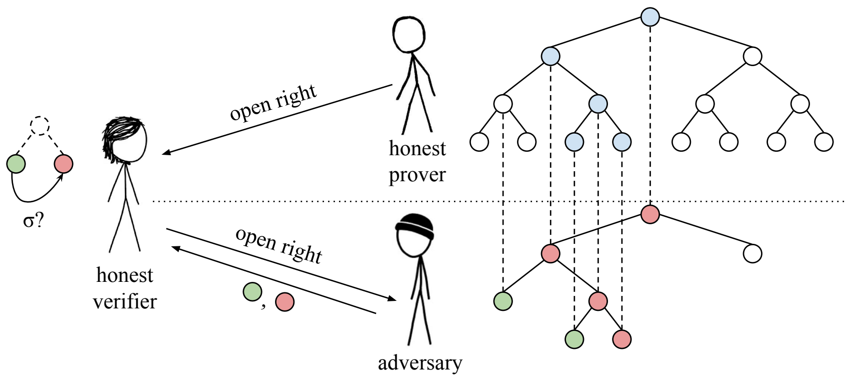


Figure 2 Honest and adversarial prover in the PoPoS bisection game (*cf.* Algs. 1, 2 and 3). The verifier iteratively requests openings of tree nodes from both provers, until the first point of disagreement is discovered.

bisection games, since at least one prover is eliminated at the end of each game. In the full version of the paper [1], we prove the security of the tournament by showing that an honest prover never loses the bisection game and an adversarial prover loses against an honest one.

Past and future. Now that the verifier obtained the state commitment signed for the most recent epoch, and confirmed its veracity, the task that remains is to discern facts about the system's state and its history. To perform queries about the current state, such as determining how much balance one owns, the verifier simply asks for Merkle inclusion proofs into the proven state commitment.

One drawback of our protocol is that the state commitment received by the verifier is the commitment at the *beginning* of the current epoch, and therefore may be somewhat stale. In order to synchronize with the latest state within the epoch, the verifier must function as a full node for the small duration of an epoch. This functionality does not harm succinctness, since epochs have a fixed, constant duration. For example, in the case of a longest-chain blockchain, the protocol works as follows. In addition to signing the state commitment, the sync committee also signs the first stable block header of its respective epoch. The block header is verified by the verifier in a similar fashion that he verified the state commitment. Subsequently, the block header can be used as a *neon genesis* block. The verifier treats the block as a replacement for the genesis block and bootstraps from there[6].

One aspect of wallets that we have not touched upon concerns the retrieval and verification of historical transactions. Consider a client that wishes to verify the inclusion of a particular historical transaction tx in the chain. Let's assume that tx is included in a block B of epoch j. This can be checked as follows. The verifier, as before, identifies the root of the correct handover tree. The verifier next asks the prover to provide him with the sync committee of epoch $j + 1$, with the corresponding inclusion proof, as well as the first stable block header B' of that epoch signed by the committee. Subsequently, he requests the short blockchain that connects B to B'. As blockchains are hash chains, this inclusion cannot be faked by an adversary.

6 Proof-of-Stake Ethereum Light Clients

The bisection games presented in Sec. 5 can be applied to a variety of PoS consensus protocols to efficiently catch up with current consensus decisions. In this section we present an instantiation for Ethereum. We also detail how to utilize the latest epoch committee to build a full-featured Ethereum JSON-RPC. This allows for existing wallets such as MetaMask to use our construction without making any changes. Our implementation can be a drop-in replacement to obtain better decentralization and performance.

Our PoPoS protocol for Ethereum does not require any changes to the consensus layer, as Ethereum already provisions for sync committees in the way we introduced in Sec. 4.

6.1 Sync Committee Essentials

Sync committees of Ethereum contain $m = 512$ validators, sampled uniformly at random from the validator set, in proportion to their stake distribution. Every sync committee is selected for the duration of a so-called *sync committee period* [23] (which we called *epoch* in our generic construction). Each period lasts 256 Ethereum epochs (these are different from our epochs), approximately 27 hours. Ethereum epochs are further divided into *slots*, during which a new block is proposed by one validator and signed by the subset of validators assigned to the slot. At each slot, each sync committee member of the corresponding period signs the block at the tip of the chain (called the *beacon chain* [23]) according to its view. The proposer of the next slot aggregates and includes within its proposal the aggregate sync committee signature on the parent block. The sync committees are determined one period in advance, and the committee for each period is contained in the block headers of the previous period. Each block also contains a commitment to the header of the last finalized block that lies on its prefix.

6.2 Linear-Complexity Light Client

Light clients use the sync committee signatures to detect the latest beacon chain block finalized by the Casper FFG finality gadget [9, 10]. At any round, the view of a light client consists of a finalized_header, the current sync committee and the next committee. The client updates its view upon receiving a LightClientUpdate object (update for short),

[6] While bootstrapping, the verifier can update the state commitment by applying the transactions within the later blocks on top of the state commitment from the neon genesis block via the function $\langle\delta\rangle$.

that contains (i) an attested_header signed by the sync committee, (ii) the corresponding aggregate BLS signature, (iii) the slot at which the aggregate signature was created, (iv) the next sync committee as stated in the attested_header, and (v) a finalized_header (called the new finalized header for clarity) to replace the one held by the client.

To validate an update, the client first checks if the aggregate signature is from a slot larger than the finalized_header in its view, and if this slot is within the current or the next period. (Updates with signatures from sync committees that are more than one period in the future are rejected.) It then verifies the inclusion of the new finalized header and the next sync committee provided by the update with respect to the state of the attested_header through Merkle inclusion proofs. Finally, it verifies the aggregate signature on the attested_header by the committee of the corresponding period. Since the signatures are either from the current period or the next one, the client knows the respective committee.

After validating the update, the client replaces its finalized_header with the new one, if the attested_header was signed by over 2/3 of the corresponding sync committee. If this header is from a higher period, the client also updates its view of the sync committees. Namely, the old next sync committee becomes the new current committee, and the next sync committee included in the attested_header is adopted as the new next sync committee.

6.3 Logarithmic Bootstrapping from Bisection Games

The construction above requires a bootstrapping light client to download at least one update per period, imposing a linear communication complexity in the life time of the chain. To reduce the communication load and complexity, the optimistic light client and superlight client constructions introduced in Secs. 4 and 5 can be applied to Ethereum.

A bootstrapping superlight client first connects to a few provers, and asks for the Merkle roots of the handover trees (*cf.* Sec. 5). The leaf of the handover tree at position j consist of all the public keys of the sync committee of period j concatenated with the period index j. If all the roots are the same, then the client accepts the sync committee at the last leaf as the most recent committee. If the roots are different, the client facilitates bisection games among conflicting provers. Upon identifying the first point of disagreement between two trees (*e.g.*, some leaf j), the client asks each prover to provide a LightClientUpdate object to justify the handover from the committee S^{j-1} to S^j. For this purpose, each prover has to provide a valid update that includes (i) an aggregate signature by 2/3 of the set S^{j-1} on an attested_header, and (ii) the set S^j as the next sync committee within the attested_header. Upon identifying the honest prover, and the correct latest sync committee, the client can ask the honest prover about the lastest update signed by the latest sync committe and containing the tip of the chain.

6.4 Superlight Client Architecture

On the completion of bootstrapping, the client has identified the latest beacon chain blockheader. The blockheader contains the commitment to the state of the Ethereum universe that results from executing all transactions since genesis up to and including the present block. Furthermore, this commitment gets verified as part of consensus. The client can perform query to the fullnode about the state of Ethereum. The result of the query can be then verified against the state commitment using Merkle inclusion proofs. This allows for the client to access the state of the Ethereum universe in a trust-minimizing way.

Fig. 3 depicts the resulting architecture of the superlight client. In today's Ethereum, a user's wallet typically speaks to Ethereum JSON-RPC endpoints provided by either a centralized infrastructure provider such as Infura or by a (trusted) Ethereum full node (could

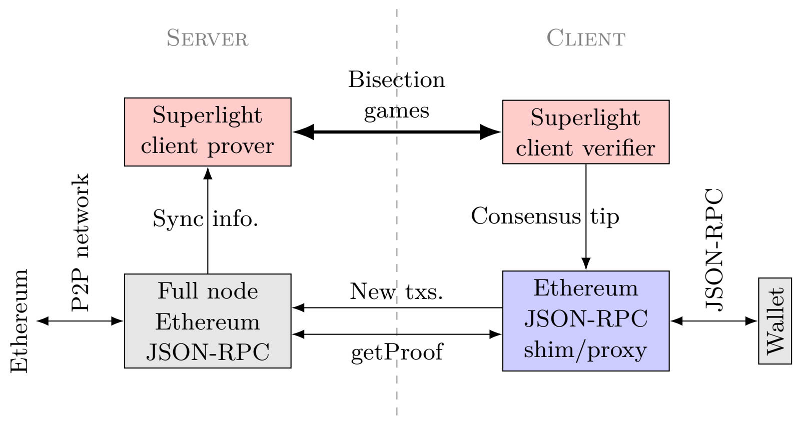


Figure 3 Ethereum superlight client architecture: On server side, an Ethereum full node feeds sync information to a bisection game prover sidecar. On client side, a bisection game verifier feeds the consensus tip into an Ethereum JSON-RPC shim/proxy, which forwards transactions coming from the wallet to the Ethereum full node, and resolves state queries with reference to the established consensus tip using Ethereum's getProof RPC endpoint.

be self-hosted). Instead, the centerpiece in a superlight client is a shim that provides RPC endpoints to the wallet, but where new transactions and queries to the Ethereum state are proxied to upstream full nodes, and query responses are verified w.r.t. a given commitment to the Ethereum state. This commitment is produced using two sidecar processes, which implement the prover and verifier of the bisection game. For this purpose, the server-side sidecar obtains the latest sync information from a full node, using what is commonly called "libp2p API". The client-side sidecar feeds the block header at the consensus tip into the shim.

7 Experiments

To assess the different bootstrapping mechanisms for Ethereum (traditional light client = TLC; optimistic light client = OLC; superlight client = SLC), we implemented them in $\approx$ 2000 lines of TypeScript code (source code available on Github[7]). We demonstrate an improvement of SLC over TLC of $9\times$ in time-to-completion, $180\times$ in communication bandwidth, and $30\times$ in energy consumption, when bootstrapping after 10 years of consensus execution. SLC improves over OLC by $3\times$ in communication bandwidth in this setting.

7.1 Setup

Our experimental scenario includes seven malicious provers, one honest prover, and a verifier. All provers run in different Heroku "`performance-m`" instances located in the "`us`" region. The verifier runs on an Amazon EC2 "`m5.large`" instance located in "`us-west-2`". The provers' Internet access is not restricted beyond the hosting provider's limits. The verifier's down- and upload bandwidth is artificially rate-limited to 100 Mbit/s and 10 Mbit/s, respectively, using "`tc`". We monitor to rule out spillover from RAM into swap space.

[7] The superlight client prototype is at `https://github.com/lightclients/poc-superlight-client`. The optimistic light client implementation is at `https://github.com/lightclients/kevlar` and `https://kevlar.sh/`. The RPC shim is at `https://github.com/lightclients/patronum`.

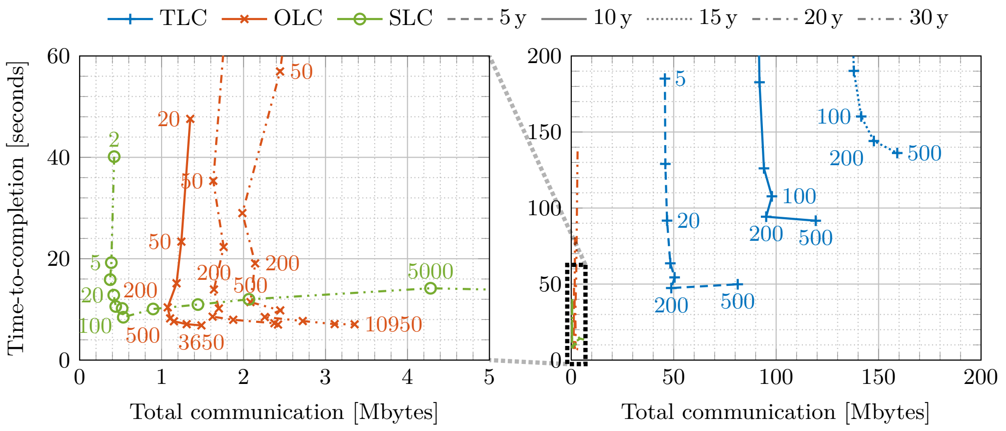


Figure 4 Time-to-completion and total communication (averaged over 5 trials) incurred by different light clients for varying internal parameters (marker labels; TLC/OLC: batch size b, SLC: Merkle tree degree d) and varying consensus execution horizon. For SLC, the curves for the considered execution horizons are virtually identical; thus, only the curve for 30 years (most challenging scenario) is shown. Pareto-optimal tradeoffs are at "tip" of resulting L-shape': for 10 years execution, at $b \approx 200$ (TLC), $b \approx 500$ (OLC), and $d \approx 100$ (SLC), respectively. OLC and SLC vastly outperform TLC, *e.g.*, for 10 years execution: 9× in time-to-completion, 180× in bandwidth. In this setting, SLC has similar time-to-completion as OLC, and 3× lower communication.

In preprocessing, we create eight valid traces of the sync committee protocol for an execution horizon of 30 years. For this purpose, we create 512 cryptographic identities per simulated day, as well as the aggregate signatures for handover from one day's sync committee to the next day's. In some experiments, we vary how much simulated time has passed since genesis, and for this purpose truncate the execution traces accordingly. One of the execution traces is used by the honest prover and understood to be the true honest execution. Adversarial provers each pick a random point in time, and splice the honest execution trace up to that point together with one of the other execution traces for the remaining execution time, *without regenerating handover signatures*, so that the resulting execution trace used by adversarial provers has invalid handover at the point of splicing. We also vary the internal parameters of the (super-)light client protocols (*i.e.*, batch size b of TLC and OLC, Merkle tree degree d of SLC).

7.2 Time-To-Completion & Total Verifier Communication

The average time-to-completion (TTC) and total communication bandwidth (TCB) required by the different light client constructions per bootstrapping occurrence is plotted in Fig. 4 for varying internal parameters (batch sizes b for TLC and OLC; Merkle tree degrees d for SLC) and varying execution horizons (from 5 to 30 years). Pareto-optimal TTC and TCB are achieved for b and d resulting "at the tip" of the "L-shaped" plot. For instance, for 10 years execution, TLC, OLC and SLC achieve Pareto-optimal TTC/TCB for $b \approx 200$, $b \approx 500$, and $d \approx 100$, respectively. Evidently, across a wide parameter range, OLC and SLC vastly outperform TLC in both metrics; *e.g.*, for 10 years execution and Pareto-optimal parameters, 9× in TTC, and 180× in TCB. In this setting, SLC has similar TTC as OLC, and 3× lower TCB (5× lower TCB for 30 years). For a closed-form expression describing the trade-off between latency and bandwidth, and the optimal choice of tree degree see Tas et al. [46].

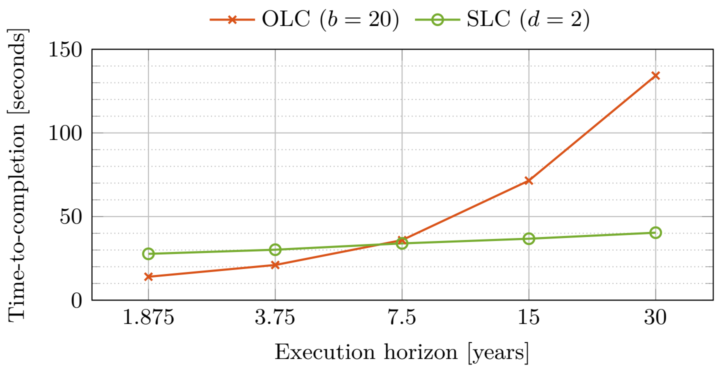


Figure 5 Time-to-completion (averaged over 5 trials) of OLC/SLC increase linearly/logarithmically with the execution horizon, respectively.

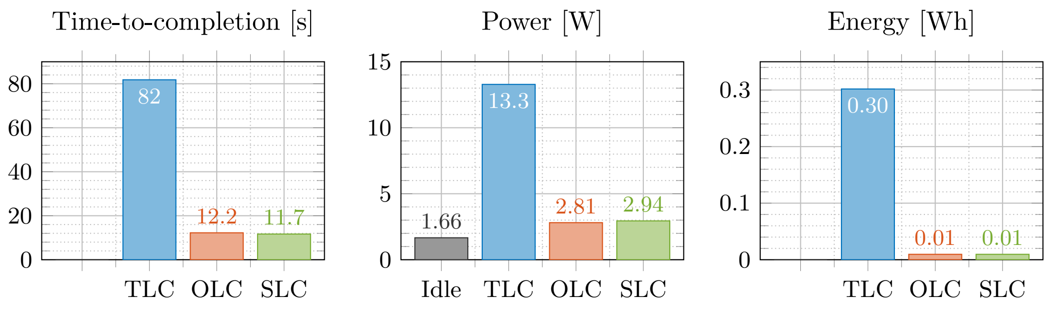


Figure 6 Energy required to bootstrap after 10 years of consensus execution using different light client constructions (averaged over 5 trials for TLC, 25 trials for OLC and SLC; internal parameters $b = 200$, $b = 500$, $d = 100$, respectively); also disaggregated into power consumption and time-to-completion. Energy required by OLC/SLC is $30\times$ lower than TLC. Contributions $\approx 4\times$ and $\approx 7\times$ can be attributed to lower power consumption and lower time-to-completion, respectively.

The fact that both TLC and OLC have TCB linear in the execution horizon, is readily apparent from Fig. 4. The linear TTC is visible for TLC, but not very pronounced for OLC, due to the concretely low proportionality constant. In comparison, SLC shows barely any dependence of TTC or TCB on the execution horizon, hinting at the (exponentially better) logarithmic dependence. To contrast the asymptotics, we plot average TTC as a function of exponentially increasing execution horizon in Fig. 5 for OLC and SLC with internal parameters $b = 20$ and $d = 2$, respectively. Note that these are not Pareto-optimal parameters, but chosen here for illustration purposes. Clearly, TTC for OLC is linear in the execution horizon (plotted in Fig. 5 on an exponential scale), while for SLC it is logarithmic.

7.3 Power & Energy Consumption

A key motivation for superlight clients is their application on resource-constrained platforms such as browsers or mobile phones. In this context, computational efficiency, and as a proxy energy efficiency, is an important metric. We ran the light clients on a battery-powered System76 Lemur Pro ("`lemp10`") laptop with Pop!_OS 22.04 LTS, and recorded the decaying battery level using "`upower`" (screen off, no other programs running, no keyboard/mouse input, WiFi connectivity; provers still on Heroku instances). From the energy consumption and wallclock time we calculated the average power consumption. As internal parameters for TLC, OLC, and SLC, we chose $b = 200$, $b = 500$, and $d = 100$, respectively (*cf.* Pareto-optimal parameters in Fig. 4). The energy required to bootstrap 10 years of consensus execution, averaged over 5 trials for TLC, and 25 trials for OLC and SLC, is plotted in Fig. 6. We disaggregate the energy consumption into power consumption and TTC for each light client, and also record the power consumption of the machine in idle. (Note, discrepancies in Figs. 4 and 6 are due to the light clients running on Amazon EC2 vs. a laptop.)

OLC and SLC have comparable TTC and power consumption, resulting in comparable energy consumption per bootstrap occurrence. The energy required by OLC and SLC is $30\times$ lower than the energy required by TLC per bootstrap occurrence (right panel in Fig. 6). This can be attributed to a $\approx 4\times$ lower power consumption (middle panel in Fig. 6) together with a $\approx 7\times$ lower TTC (left panel in Fig. 6). The considerably lower energy/power consumption of OLC/SLC compared to TLC is due to the lower number of signature verifications (and thus lower computational burden). Note that a sizeable fraction of OLC's/SLC's power consumption can be attributed to system idle (middle panel in Fig. 6). When comparing light clients in terms of *excess* energy consumption (*i.e.*, subtracting idle consumption) per bootstrapping, then OLC and SLC improve over SLC by $64\times$.

8 Analysis

The theorems for succinctness and security of the PoPoS protocol are provided below. Proofs are in the full version of this paper [1]. Security consists of two components: completeness and soundness.

▶ **Theorem 4** (Succinctness). *Consider a verifier that invokes a bisection game at round r between two provers that provided different handover tree roots. Then, the game ends in $O(\log(r))$ steps of interactivity and has a total communication complexity of $O(\log(r))$.*

▶ **Theorem 5** (Completeness). *Consider a verifier that invokes a bisection game at round r between two provers that provided different handover tree roots. Suppose one of the provers is honest. Then, the honest prover wins the bisection game.*

▶ **Theorem 6** (Soundness). *Let H^s be a collision resistant hash function. Consider a verifier that invokes a bisection game executed at round r of a secure underlying PoS protocol between two provers that provided different handover tree roots. Suppose one of the provers is honest, and the signature scheme satisfies existential unforgeability. Then, for all PPT adversarial provers $\mathcal{A}$, the prover $\mathcal{A}$ loses the bisection game against the honest prover with overwhelming probability in λ.*

▶ **Theorem 7** (Tournament Runtime). *Consider a tournament ran at round r with $|\mathcal{P}|$ provers one of which is honest. The tournament ends in $O(|\mathcal{P}|\log(r))$ steps of interactivity, and has total communication complexity $O(|\mathcal{P}|\log(r))$.*

▶ **Theorem 8** (Security). *Let H^s be a collision resistant hash function. Consider a tournament executed between an honest verifier and $|\mathcal{P}|$ provers at round r. Suppose one of the provers is honest, the signature scheme satisfies existential unforgeability, and the PoS protocol is secure. Then, for all PPT adversaries $\mathcal{A}$, the state commitment obtained by the verifier at the end of the tournament satisfies state security with overwhelming probability in λ.*

References

1 Shresth Agrawal, Joachim Neu, Ertem Nusret Tas, and Dionysis Zindros. Proofs of proof-of-stake with sublinear complexity. Cryptology ePrint Archive, Paper 2022/1642, 2022. URL: `https://eprint.iacr.org/2022/1642`.

2 Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains, 2014. URL: `https://blockstream.com/sidechains.pdf`.

3 Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS*, pages 913–930. ACM, 2018.

4 Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Paper 2020/352, 2020. URL: `https://eprint.iacr.org/2020/352`.

5 Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus, 2018. `arXiv:1807.04938v3`.

6 Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. In *IEEE Symposium on Security and Privacy*, pages 928–946. IEEE, 2020.

7 Vitalik Buterin. A next-generation smart contract and decentralized application platform, 2014.

8 Vitalik Buterin. Proof of Stake: How I Learned to Love Weak Subjectivity, November 2014. URL: `https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/`.

9 Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget, 2017. `arXiv:1710.09437v4`.

10 Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining ghost and casper, 2020. `arXiv:2003.03052v3`.

11 Ran Canetti, Ben Riva, and Guy N. Rothblum. Practical delegation of computation using multiple servers. In *CCS*, pages 445–454. ACM, 2011.

12 Ran Canetti, Ben Riva, and Guy N. Rothblum. Refereed delegation of computation. *Inf. Comput.*, 226:16–36, 2013.

13 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186. USENIX Association, 1999.

14 Pyrros Chaidos and Aggelos Kiayias. Mithril: Stake-based threshold multisignatures. Cryptology ePrint Archive, Paper 2021/916, 2021. URL: `https://eprint.iacr.org/2021/916`.

15 Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. Sok: Blockchain light clients. In *Financial Cryptography*, volume 13411 of *LNCS*, pages 615–641. Springer, 2022.

16 ConsenSys. MetaMask Surpasses 10 Million MAUs, Making It The World's Leading Non-Custodial Crypto Wallet, August 2021. URL: `https://consensys.net/blog/press-release/metamask-surpasses-10-million-maus-making-it-the-worlds-leading-non-custodial-crypto-wallet/`.

17 Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. Cryptology ePrint Archive, Paper 2016/919, 2016. URL: `https://eprint.iacr.org/2016/919`.

18 Stelios Daveas, Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. A gas-efficient superlight bitcoin client in solidity. In *AFT*, pages 132–144. ACM, 2020.

19 Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)*, volume 10821 of *LNCS*, pages 66–98. Springer, 2018.

20 Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019.

21 Grin Developers. Merkle Mountain Ranges (MMR). URL: `https://docs.grin.mw/wiki/chain-state/merkle-mountain-range/`.

22 Ethereum Developers. Altair Light Client – Light Client, 2023. URL: `https://github.com/ethereum/consensus-specs/blob/5c64a2047af9315db4ce3bd0eec0d81194311e46/specs/altair/light-client/light-client.md`.

23 Ethereum Developers. Altair Light Client – Sync Protocol, 2023. URL: `https://github.com/ethereum/consensus-specs/blob/e9f1d56807d52aa7425f10160a45cb522345468b/specs/altair/light-client/sync-protocol.md`.

24 Ariel Gabizon, Kobi Gurkan, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, Michael Straka, Eran Tromer, and Psi Vesely. Plumo: Towards scalable interoperable blockchains using ultra light validation systems, 2020. URL: `https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-plumo_celolightclient.pdf`.

25 Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. Cryptology ePrint Archive, Paper 2014/765, 2014. URL: `https://eprint.iacr.org/2014/765`.

26 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, volume 9057 of *LNCS*, pages 281–310. Springer, 2015.

27 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO (1)*, volume 10401 of *LNCS*, pages 291–323. Springer, 2017.

28 Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *IEEE Symposium on Security and Privacy*, pages 139–156. IEEE, 2019.

29 Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In *CRYPTO*, volume 2139 of *LNCS*, pages 332–354. Springer, 2001.

30 Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*, pages 1353–1370. USENIX Association, 2018.

31 Kostis Karantias. Sok: A taxonomy of cryptocurrency wallets. Cryptology ePrint Archive, Paper 2020/868, 2020. URL: `https://eprint.iacr.org/2020/868`.

32 Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Compact storage of superblocks for nipopow applications. In *MARBLE*, pages 77–91. Springer, 2019.

33 Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Proof-of-burn. In *Financial Cryptography*, volume 12059 of *LNCS*, pages 523–540. Springer, 2020.

34 Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

35 Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In *Financial Cryptography Workshops*, volume 9604 of *LNCS*, pages 61–78. Springer, 2016.

36 Aggelos Kiayias, Nikos Leonardos, and Dionysis Zindros. Mining in logarithmic space. In *CCS*, pages 3487–3501. ACM, 2021.

37 Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. In *Financial Cryptography*, volume 12059 of *LNCS*, pages 505–522. Springer, 2020.

38 Aggelos Kiayias, Andrianna Polydouri, and Dionysis Zindros. The velvet path to superlight blockchain clients. In *AFT*, pages 205–218. ACM, 2021.

39 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO (1)*, volume 10401 of *LNCS*, pages 357–388. Springer, 2017.

40 Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *Financial Cryptography Workshops*, volume 11599 of *LNCS*, pages 21–34. Springer, 2019.

41 Jae Kwon and Ethan Buchman. A network of distributed ledgers – cosmos whitepaper. URL: `https://v1.cosmos.network/resources/whitepaper`.

42 Rongjian Lan, Ganesha Upadhyaya, Stephen Tse, and Mahdi Zamani. Horizon: A gas-efficient, trustless bridge for cross-chain transactions, 2021. `arXiv:2101.06000v1`.

43 Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, volume 293 of *LNCS*, pages 369–378. Springer, 1987.

44 Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT (2)*, volume 10211 of *LNCS*, pages 643–673, 2017.

45 Succinct Labs. Building the end game of interoperability with zkSNARKs, 2023. URL: `https://www.succinct.xyz/`.

46 Ertem Nusret Tas, Dionysis Zindros, Lei Yang, and David Tse. Light clients for lazy blockchains. Cryptology ePrint Archive, Paper 2022/384, 2022. URL: `https://eprint.iacr.org/2022/384`.

47 Peter Todd. Merkle mountain ranges, October 2012. URL: `https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md`.

48 Jason Wise. Metamask Statistics 2023: How Many People Use Metamask?, March 2023. URL: `https://earthweb.com/metamask-statistics/`.

49 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2014.

50 Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In *CCS*, pages 3003–3017. ACM, 2022.

51 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019.

52 Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. Sok: Communication across distributed ledgers. In *Financial Cryptography (2)*, volume 12675 of *LNCS*, pages 3–36. Springer, 2021.

53 Alexei Zamyatin, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar R. Weippl, and William J. Knottenbelt. A wild velvet fork appears! inclusive blockchain protocol changes in practice - (short paper). In *Financial Cryptography Workshops*, volume 10958 of *LNCS*, pages 31–42. Springer, 2018.

54 Maksym Zavershynskyi. ETH-NEAR Rainbow Bridge, August 2020. URL: `https://near.org/blog/eth-near-rainbow-bridge/`.

Condorcet Attack Against Fair Transaction Ordering

Mohammad Amin Vafadar
University of Alberta, Edmonton, Canada

Majid Khabbazian
University of Alberta, Edmonton, Canada

Abstract

We introduce the *Condorcet attack*, a new threat to fair transaction ordering. Specifically, the attack undermines batch-order-fairness, the strongest notion of transaction fair ordering proposed to date. The batch-order-fairness guarantees that a transaction `tx` is ordered before `tx′` if a majority of nodes in the system receive `tx` before `tx′`; the only exception (due to an impossibility result) is when `tx` and `tx′` fall into a so-called "Condorcet cycle". When this happens, `tx` and `tx′` along with other transactions within the cycle are placed in a batch, and any unfairness inside a batch is ignored.

In the Condorcet attack, an adversary attempts to undermine the system's fairness by imposing Condorcet cycles to the system. In this work, we show that the adversary can indeed impose a Condorcet cycle by submitting as few as two otherwise legitimate transactions to the system. Remarkably, the adversary (e.g., a malicious client) can achieve this even when all the nodes in the system behave honestly. A notable feature of the attack is that it is capable of "trapping" transactions that do not naturally fall inside a cycle, i.e. those that are transmitted at significantly different times (with respect to the network latency). To mitigate the attack, we propose three methods based on three different complementary approaches. We show the effectiveness of the proposed mitigation methods through simulations, and explain their limitations.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Transaction ordering, fairness, Condorcet cycle

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.15

1 Introduction

The first blockchain application, Bitcoin, emerged in the midst of the financial crisis of 2008, caused in part by the excessive trust placed in centralized institutions. Blockchain technology changed this. In blockchain, there is no central authority or intermediary controlling the entire system. Instead, transactions are validated and included through a consensus mechanism among the participating parties. Decentralization also promotes transparency and reduces the possibility of fraud or corruption since all transactions are publicly recorded and visible to all participants on the network.

Despite the decentralized nature of blockchain systems, the ordering of transactions is carried out in a centralized manner; the miner/validator who creates a block determines the ordering of transactions within the block. This gives too much power to a single entity as the success and profitability of a transaction can be determined by the order in which the transaction appears in a block [6, 1, 8, 9, 16]. For instance, when a Non-Fungible Token (NFT) is dropped in a given block, transactions positioned earlier in the block have a higher chance of acquiring the NFT compared to those placed later.

To address this issue, several existing works [12, 20, 11, 4, 10, 13] proposed decentralized methods for handling transaction ordering, where instead of a single node, a committee of nodes collectively decide on the ordering of received transactions. At the core of these methods, each node in the system reports a list of transactions in the order the node has received them. The system then generates and agrees on a "fair" ordering by taking the reported orderings into account.

Finding a fair ordering is not trivial. For instance, suppose that for any two transactions $\texttt{tx}_1$ and $\texttt{tx}_2$, we require $\texttt{tx}_1$ to be placed before $\texttt{tx}_2$ if a large majority of nodes in the system claim to have received $\texttt{tx}_1$ before $\texttt{tx}_2$. Despite being a primitive requirement, no method can provide a guarantee due to an impossibility result rooted in social choice theory [2]. As an example, consider a system consisting of three nodes, where each node has received three transactions: $\texttt{tx}_1$, $\texttt{tx}_2$, and $\texttt{tx}_3$. Suppose the nodes report the ordering as $[\texttt{tx}_1, \texttt{tx}_2, \texttt{tx}_3]$, $[\texttt{tx}_2, \texttt{tx}_3, \texttt{tx}_1]$, and $[\texttt{tx}_3, \texttt{tx}_1, \texttt{tx}_2]$. In this case, $\texttt{tx}_1$ is reported to be before $\texttt{tx}_2$ by two nodes (i.e. the majority), $\texttt{tx}_2$ is reported to be before $\texttt{tx}_3$ by two nodes, and $\texttt{tx}_3$ is reported to be before $\texttt{tx}_1$ by two nodes. This essentially creates a cycle, referred to as *Condorcet cycle* [5], which prevents any final ordering from respecting the views of the majority on how transactions should be ordered.

The existing fair ordering methods adopt a relaxed approach to ordering transactions inside a Condorcet cycle. For instance, Cachin et al. in Quick-Fairness [4] do not mention any ordering mechanism for such transactions, and Kelkar et al. in Aequitas [12] suggest a simple alphabetical ordering. This relaxed approach is, perhaps, due to two reasons: 1) it is not possible to guarantee fair ordering of transactions inside a cycle; 2) Condorcet cycles occur infrequently in practice, and when they do occur, they usually involve transactions that are received around the same time by the nodes in the system. Nevertheless, in this work, we show that Condorcet cycles deserve more attention as they can be created "artificially" by adversaries through what we refer to as the *Condorcet attack*. An interesting feature of the Condorcet attack proposed in this work is that it can be conducted by a client outside the system. In particular, the attack can be effectively executed even when all the nodes in the system are honest!

As will be explained later, in the Condorcet attack, an adversarial client sends a small number of transactions to different nodes in the system. The adversary chooses the timing and order of these transactions to create a Condorcet cycle that traps many honest transactions in it (a Condorcet cycle with only the adversary's transactions in it is all but harmless to the system.). This cycle has to be broken by the leader in a leader-based method in order to establish a total ordering. Even if the leader is honest, the act of breaking the cycle could change the order of honest transactions, which would have otherwise been appropriately ordered[1].

Defending against the Condorcet attack is not straightforward. It is partly because it is challenging to differentiate between honest transactions and otherwise valid transactions that are submitted with the intention of creating a cycle. It becomes notably more challenging to safeguard the system when, in addition to the adversarial client outside the system, the leader and possibly a fraction of the nodes in the system are adversarial. Nevertheless, in this work, we propose three mitigation techniques based on three different approaches. The proposed techniques complement each other and can work together harmoniously to maximize resistance against the attack.

In summary, we make the following contributions. We introduce a framework for a new type of attack (Condorcet attack) against fair transaction ordering. We show that the attack can be highly successful in trapping honest transactions in a cycle. To mitigate the attack, we propose three techniques based on three different complimentary approaches, and show the effectiveness of the technique through simulations.

[1] Kelkar et al. [11] consider it a success for an adversary if the adversary places two transactions into the same cycle when they should not have been.

2 Related Work

The classical approach to mandating fair transaction ordering is through *secure causal ordering*, a method introduced by Birman and Reiter in 1994 [17], and later improved by Cachin et al. in 2001 [3]. This method uses encryption to conceal the content of transactions during the ordering process, and allows decryption of transactions only after the order of transactions is finalized. This prevents an adversary from observing the content of transactions during the ordering process, thereby effectively eliminating attacks such as the sandwich attack [16] that rely on inspecting transaction contents. However, the method is unable to prevent "blind front-running attacks" where, for instance, the adversary's sole objective is to order her transaction first (to, for example, get priority in purchasing a token). In addition, the method cannot prevent attacks based on transactions' metadata, as metadata (such as the source of a transaction) is not encrypted.

The second approach to mandating fair transaction ordering involves a first-come, first-served strategy. This approach is complementary to the first approach and has been the focus of several recent studies. The existing methods that follow this strategy can be broadly classified into two categories: timestamp-based methods and batch-based methods. Timestamp-based methods are computationally inexpensive but require synchronized clocks. Batch-based methods, on the other hand, offer stronger fairness than timestamp-based methods, but can tolerate fewer adversarial nodes.

Timestamp-based Methods. An example of a timestamp-based protocol is Pompe [20] due to Zhang et al. Pompe introduces a notion of fairness called the *ordering linearizability*. This notion stipulates that if the highest timestamp of a transaction `tx` is less than the lowest timestamp of a transaction $\texttt{tx}'$ among honest nodes, then `tx` must be ordered before $\texttt{tx}'$ in the final order of transactions. Although it can enforce ordering linearizability, Pompe suffers from censorship issues, as noted in [11].

Kursawe's Wendy protocol [13] is another timestamp-based protocol that defines a notion of fairness called *timed-relative-fairness*. This notion requires that if all honest nodes received a transaction `tx` before time τ, and transaction $\texttt{tx}'$ after τ, then `tx` must be ordered before $\texttt{tx}'$.

Batch-based Methods. Aequitas [12] by Kelkar et al. is a batch-based method proposed for fair transaction ordering. Aequitas enforces a fairness notion known as the *γ-batch-order-fairness*. The notion requires that if two transactions `tx` and $\texttt{tx}'$ are received by all nodes in a system with n nodes, and γn nodes received `tx` before $\texttt{tx}'$, then all honest nodes must output `tx` no later than $\texttt{tx}'$. Aequitas suffers from high communication complexity of $\mathcal{O}(n^3)$, and can guarantee only a weak notion of liveness, one of the two pillars of consensus security.

The second batch-based method is Quick-Fairness [4] proposed by Cachin et al. This method enforces a fairness notion called the *κ-differential order-fairness*. This notion mandates that if the number of nodes that have received transaction `tx` before $\texttt{tx}'$ exceeds $\kappa + 2f$ for some $\kappa \geq 0$, then `tx` should be ordered no later than $\texttt{tx}'$, where f is the maximum number of adversarial nodes in the system. Kelkar et al. [11] show that this notion of fairness is indeed a re-parameterized version of the γ-batch-order-fairness notion. They also demonstrate that the Quick-Fairness protocol satisfies fairness only when all nodes are honest.

Kelkar et al. addressed the shortcomings of Aequitas in their protocol called Themis [11]. Themis satisfies the γ-batch-order-fairness notion, and solves the liveness problem of Aequitas. Moreover, SNARK-Themis variant offers a communication complexity of $\mathcal{O}(n)$ and standard

Themis offers a communication complexity of $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^3)$ offered by Aequitas. In addition, it satisfies a more generalized notion of fairness than the one used in Quick-Fairness and a stronger notion of fairness than those used in the existing time-based methods. For these reasons, in our work, we focus on Themis and Aequitas as the state-of-the-art fair transaction ordering methods.

3 Model

System. We consider a permissioned system with a committee of n nodes. The nodes receive transactions directly from clients, and submit the list of their received transactions together with the order in which the transactions were received to a special node called the leader. The leader collects the lists of transactions from the nodes, and proposes a final ordering using a pre-decided fair-ordering protocol. The leader in the system is not fixed, and can change through a pre-determined protocol.

Fair Ordering. We adopt the batch-order-fairness from [12, 11], the strongest notion of fair ordering proposed to date. For a parameter $\frac{1}{2} < \gamma \leq 1$, the batch-order-fairness specifies that if a fraction γ of nodes receive a transaction $\mathtt{tx}$ before receiving another transaction $\mathtt{tx}'$, then $\mathtt{tx}$ must be placed in the order before $\mathtt{tx}'$, with exceptions allowed only if $\mathtt{tx}$ and $\mathtt{tx}'$ are within the same Condorcet cycle (Condorcet cycles are defined in Section 4). Transactions within a cycle are placed in a batch, and are ordered by a method that we refer to as *batch-ordering scheme*. The existing fair ordering protocols either do not specify a batch-ordering scheme or propose a simple one (e.g., an alphabetical-based scheme [12]).

Network. The network utilizes public key infrastructure and secure digital signatures for communications. As in [12], we consider two networks: the (standard) internal network (for communication amongst nodes in the system) and the external network (for clients to transmit their transactions to the system).

We assume that the network operates under partial synchrony [7], meaning that there is a network delay Δ (not known to the nodes) that limits the amount of time it takes for messages to be delivered between nodes.

Adversary. We consider an adversary who has control over $f \geq 0$ out of n nodes, and also possesses at least one client capable of submitting transactions to the system. The adversary can deviate arbitrarily from the protocol. The adversary does not have control over the external network, but may have full control over the internal network, hence can delay and reorder messages up to the bound Δ.

4 Preliminaries

Graph Terminology. We use $G = (V, E)$ to denote a graph with the set of vertices V and the set of edges E. In this work, each vertex represents a transaction, therefore, we use the terms vertices and transactions interchangeably. Unless otherwise specified, we use an unweighted and directed graph. In the case of a weighted graph, the weight or cost associated with the edge $(u, v) \in E$ is represented by $w(u, v)$.

A *tournament graph* is a directed graph where every pair of distinct vertices is connected by a directed edge in either of two possible directions. A *Strongly Connected Component (SCC)* in a graph is a maximal subgraph in which there is a path from every vertex to every other vertex. A *condensation graph* is obtained from the original graph by combining its SCCs into a single vertex. A *Directed Acyclic Graph (DAG)* is a directed graph that contains no cycles, meaning it is possible to move from one vertex to another along the directed edges, but it is not possible to return to the original vertex by following a sequence of directed edges. A *topological sort* is an ordering of the vertices in a DAG such that for every directed edge (u, v), vertex u appears before vertex v. In other words, if there is a directed edge from vertex u to vertex v, then u must appear before v in the topological sort. A *Hamiltonian Path* is a path in a graph that passes through every vertex exactly once. A *Hamiltonian Cycle* is a cycle in a graph that passes through every vertex exactly once.

Themis. Themis is the latest ordering method which achieves batch-order-fairness in the presence of an adversary who controls up to $f < \frac{(2\gamma-1)n}{4}$ nodes out of n nodes. Themis categorized received transactions into three different categories.

- Solid Transactions: A transaction is solid if it has been received by at least $n - 2f$ nodes. A solid transaction is one that has been received by enough honest nodes that the leader can unambiguously include it in the current proposal while respecting the fairness guarantees.
- Blank Transactions: A transaction is blank if it has not been received by at least $n(1 - \gamma) + f + 1$ nodes. A blank transaction has not been received by enough nodes yet, hence excluding it from the current proposal will not violate fairness with respect to transactions that are included.
- Shaded Transactions: A shaded transaction is a transaction that is neither solid nor blank. A shaded transactions is received by enough nodes to be included to preserve fairness, but not enough nodes to finalize its position in the current proposal.

Themis is a leader-based method and works in three phases, as described below.

- Phase 1 (Fair Propose): The Fair Propose phase is the first phase of the algorithm, where each node proposes a set of transactions and their local orderings to the leader. The leader then uses the local orderings of $n - f$ nodes to build a dependency graph. In the dependency graph, an edge from a vertex v_1 to v_2 indicates that the transaction v_1 should be placed before the transaction v_2. From the dependency graph, the leader then computes the condensation graph and its topological sorting to output a fair ordering.
- Phase 2 (Fair Update): The Fair Update phase is the second phase of the algorithm, where the leader node updates the ordering for previous proposals. This is necessary since this is part of the deferred ordering technique, and new transactions may depend on previously proposed transactions, and these dependencies need to be accounted for in the ordering. The Fair Update algorithm takes the local transaction orderings of $n - f$ nodes for previously proposed shaded transactions as input and outputs the updated dependencies.
- Phase 3 (Fair Finalize): The Fair Finalize phase is the third and final phase of the algorithm, where a sequence of proposals is finalized into a final ordering. The Fair Finalize algorithm updates the graphs for each proposal and computes the condensation graphs and their topological sorting. It then retrieves the final transaction ordering for each proposal based on the Hamiltonian cycles of the vertices in the sorted condensation graphs.

Condorcet Cycles. As mentioned above, Themis constructs a dependency graph, a directed graph where each vertex represents a transaction, and an edge from a vertex v_1 to v_2 indicates that the transaction corresponding to v_1 should be placed before the transaction corresponding to v_2. We refer to any cycle in this dependency graph as a *Condorcet cycle*. We note that cycles can occur in a dependency graph because of the Condorcet paradox [11].

5 Condorcet Attack

In this section, we present the framework of the Condorcet attack. The attack aims at trapping honest transactions (i.e., transactions submitted by honest clients) inside a Condorcet cycle. If there is no effective batch-ordering scheme in place (e.g., if the batch-ordering scheme is alphabetical-based as suggested in [12]), this can change the ordering of the honest transactions even when all the nodes in the system are honest.

An adversary can take different strategies to impose a Condorcet cycle. For instance, suppose that the adversary controls f nodes, including the leader, in the system. The adversary then controls f local orderings, and can manipulate these orderings in a way to create a cycle. In the simulation section, we show that this strategy can not only create a cycle but also chain the cycles to involve more honest transactions. Nevertheless, the length of these cycles is typically small and the chain usually breaks rather quickly. As a result, this strategy is not effective in trapping distant transactions[2] (e.g., two transactions whose times of submission are separated by a multiple of the average network latency).

Another strategy, which is the one we take in this work, is to create a Condorcet cycle by injecting (valid) transactions into the system following a pre-described pattern. This can be done by an adversarial client outside the system, and can be effective even when all the nodes in the system are honest. The attack will be more effective in creating cycles and bypassing potential countermeasures if the adversary controls a fraction of nodes in the system (see Example 5).

The immediate damage of imposing a Condorcet cycle, as mentioned earlier, is that it can change the true ordering of honest transactions. In addition to this, the attack may be used to conduct other malicious activities; for instance, the adversary can create a cycle and then with the help of an adversarial leader can try to place its own transaction in desired positions in the final ordering.

▶ **Example 1.** Let $\mathcal{P} = \{P_1, P_2, P_3\}$ be a partition of nodes, where P_1, P_2 and P_3 are three parts with almost equal size. In this simple example, the adversary $\mathcal{C}$ uses/injects two transactions $\mathtt{A}, \mathtt{B}$ (i.e., $\mathcal{S} = \{\mathtt{A}, \mathtt{B}\}$). In the initialization phase, $\mathcal{C}$ sends the transaction `A` and then `B` to all the nodes in part P_1, and sends the transaction `B` to all the nodes in part P_2 (it sends no transactions to the nodes in part P_3). Then, after the pause period, $\mathcal{C}$ sends `A` to all the nodes in part P_2, and transaction `A` and `B`, in that order, to all the nodes in part P_3. Suppose that during the pause phase, the nodes receive three honest transactions $\mathtt{tx}_1$, $\mathtt{tx}_2$, and $\mathtt{tx}_3$ all the in that order. The local ordering of transactions at each node will be then:

$$\begin{aligned} P_1 &: \quad [\mathtt{A}, \mathtt{B}, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3] \\ P_2 &: \quad [\mathtt{B}, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}] \\ P_3 &: \quad [\mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}, \mathtt{B}] \end{aligned}$$

[2] The analysis of why this occurs is left for future work.

Note that without the adversarial client $\mathcal{C}$ disturbing the system (i.e., without transactions `A` and `B`), the system would have had an easy job of ordering the honest transactions as all the nodes in the system have received the honest transactions in the same order, i.e. [$\texttt{tx}_1$, $\texttt{tx}_2$, $\texttt{tx}_3$]. Because of the adversary's transactions `A`, `B`, and `C`, however, we have a cycle now as illustrated in Figure 1. In this figure, an edge from a transaction `tx` to a transaction `tx'` indicates that the majority of the nodes have received `tx` before `tx'`.

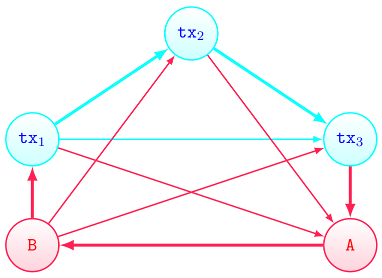


Figure 1 A Condorcet cycle created using two transactions `A` and `B`.

Attack Framework. In this section, we provide a general construction that encompasses the different variants of the Condorcet attack. Let $\mathcal{C}$ be a client controlled by the adversary, and $\mathcal{S}$ be a set of arbitrary but valid transactions created by $\mathcal{C}$. Let $\mathcal{P}$ be a partition of the nodes in the system. In its general form, the Condorcet attack is executed in three phases:

- Phase 1 (Initialization): In this phase, the client $\mathcal{C}$ sends a number of transactions from the set $\mathcal{S}$ to each node in the system. The set of transactions sent to a node can be different from that sent to another node. More specifically, the client $\mathcal{C}$ assigns a subset $\mathcal{S}_i$ of $\mathcal{S}$ (possibly an empty subset) to each part $\mathcal{P}_i$ in the partition $\mathcal{P}$. It then determines an ordering for each subset $\mathcal{S}_i$, and sends the transactions in $\mathcal{S}_i$ to all the nodes in part $\mathcal{P}_i$ with the determined order.
- Phase 2 (Pause): In the second phase, the attacker waits for a specific amount of time, referred to as the pause time, for the honest transactions to be received by the nodes. The adversary can trap more transactions within a cycle as the pause time increases. However, the pause time should be limited to a single consensus round in the system as the attack should not extend across multiple consensus rounds.
- Phase 3 (Finalization): The third and final phase is the finalization phase, where the attacker completes the Condorcet cycle by sending a new set of transactions to each part in the partition. More specifically, the client $\mathcal{C}$ assigns a subset $\mathcal{S}'_i$ of $\mathcal{S}$ (typically a different subset than $\mathcal{S}_i$, used in the initialization phase) to each part $\mathcal{P}_i$ in the partition $\mathcal{P}$. It then determines an ordering for each subset $\mathcal{S}'_i$, and sends the transactions in $\mathcal{S}'_i$ to all the nodes in part $\mathcal{P}_i$ with the determined order.

▶ Remark 2. In practice, nodes in the system may receive some honest transactions during the initialization and/or finalization phases. These transactions may or may not get trapped in the Condorcet cycle. Based on our simulation results, however, the vast majority of honest solid transactions during the pause time fall into the Condorcet cycle.

▶ Remark 3. A potential issue that can impact the success of the Condorcet attack is that the external network may deliver the transactions injected by the adversary out of order. For instance, in Example 1, the transactions `A` and `B` may be received out of order by the nodes in part P_1, in which case a cycle does not occur. If the network is prone to packet reordering, then to improve its success, the adversary can execute multiple Condorcet attacks concurrently through what we refer to as *cloning*.

Cloning. Packet reordering can happen in a network because of various factors such as network congestion, routing algorithms, and the physical distance between the source and the destination. To conduct a successful Condorcet attack, it is important that nodes receive the injected packets in the order they were transmitted; a deviation from the intended order may result in the failure of the attack.

To increase the success probability of the attack in the presence of network reordering, the adversary can send cloned transactions to the nodes: Instead of sending a single transaction $\mathtt{A}$, the adversary sends multiple clones of the transaction. For instance, in Example 1, the adversary can send $\mathtt{A}_1$ and $\mathtt{A}_2$ instead of $\mathtt{A}$, and sends $\mathtt{B}_1$ and $\mathtt{B}_2$ instead of $\mathtt{B}$. Essentially, the adversary interleaves the execution of two Condorcet attacks (for better results, the adversary can interleave several instances of the attack). Then, if the network does not change the order of the transactions, the nodes in parts P_1, P_2, and P_3 will receive transactions as follows:

$$\begin{aligned}
P_1 :&\quad [\mathtt{A}_1, \mathtt{A}_2, \mathtt{B}_1, \mathtt{B}_2, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3]\\
P_2 :&\quad [\mathtt{B}_1, \mathtt{B}_2, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}_1, \mathtt{A}_2]\\
P_3 :&\quad [\mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}_1, \mathtt{A}_2, \mathtt{B}_1, \mathtt{B}_2]
\end{aligned}$$

In Section 7.3, we show that cloning can significantly increase the success rate of the Condorcet attack in the presence of network reordering.

Impact on Current Solutions. The current fair transaction ordering protocols either do not offer a batch-ordering scheme (e.g. [4]) or offer a primitive one (e.g. [12]). For instance, the proposed batch-ordering scheme in Aequitas [12] is alphabetical ordering. Therefore, if an adversary creates a Condorcet cycle, as in Example 1, the honest transactions will be ordered alphabetically rather than by the time of their arrival.

Themis [11], proposes a more thoughtful batch-ordering scheme. In this scheme, a Hamiltonian cycle is built and then used to order transactions in the cycle. The latest version of Themis at the time of writing this work suggests to break the weakest link in the Hamiltonian cycle in order to convert it into a Hamiltonian path. We use this version of Themis in our work. In the best-case scenario, the order of honest transactions in the Hamiltonian cycle is preserved. Even in this case, the final ordering of these transactions can change because the Hamiltonian cycle has to be converted into a path by breaking the cycle at one point. It is at this point where honest transactions can be divided into two groups. The ordering of the honest transactions within each group remains correct, but the ordering of any two transactions from different groups will be incorrect. Therefore, similar to [4] and [12], Themis is vulnerable to the Condorcet attack even if all the nodes (including the leader) in the system are honest.

To combat the Condorcet attack, a natural approach is to use a strong batch-ordering scheme. For instance, in Example 1, we can observe that all the nodes report $\mathtt{tx}_1$ before $\mathtt{tx}_2$, and all the nodes report $\mathtt{tx}_2$ before $\mathtt{tx}_3$, whereas only two third of the nodes report $\mathtt{A}$ before $\mathtt{B}$. In this example, the weakest link is between adversarial transactions, and breaking it (as suggested by Themis) does not change the true ordering of the honest transactions. This solution works for the scenario described in Example 1. However, this solution may not work in other settings, for example when the adversary controls a faction of nodes in the system (see Example 5).

6 Mitigation

Despite its simplicity, it is not straightforward to completely defeat the Condorcet attack. In the following, we present three mitigation techniques based on three different approaches to hinder an adversary from successfully executing the attack. We elaborate on the strength of each technique and confirm it through simulations later in Section 7. We also explain the limitation of each technique, i.e. under what settings/assumptions the technique may not be effective.

An interesting feature of the proposed mitigation methods is that they do not conflict with each other, thus in practice, they can be applied together for the maximum defense against the attack. Another interesting feature of the proposed mitigation methods is that they can be easily applied to Themis, which is currently the strongest fair-ordering solution in the literature. We elaborate on this when we cover each proposed mitigation.

6.1 Ranked Pairs Batch-ordering

The approach we take in our first proposed mitigation is to use a strong batch-ordering scheme to order transactions within a batch. Formally, a batch-ordering scheme is a method that takes as input a strongly connected (possibly weighted) directed graph $G = (V, E)$, and returns an ordering of the vertices V. The strongly connected graph represents the transactions that are in a batch/cycle.

The candidate for our batch-ordering scheme is *ranked pairs*, an electoral system developed by Nicolaus Tideman in 1987 [19]. Ranked pairs satisfies many natural and well-studied axiomatic properties in social choice theory[3] and is resistant to certain manipulations including adding, deleting and changing a fraction of orderings reported by nodes [15]. In ranked pairs, the ordering is essentially achieved by choosing a maximal subset E' of E in the inputted graph $G = (V, E)$ with high weights such that $G' = (V, E')$ is a DAG. The DAG is then used to establish an ordering of the vertices V.

More specifically, our ranked pairs batch-ordering scheme takes as input a weighted directed graph $G = (V, E)$. Let $E_1 = E$. In step i, $i \geq 1$, the algorithm selects an edge $(u, v) \in E_i$ with the highest weight[4]. It then sets the order $u \prec v$, unless this violates the transitivity of the orders decided in previous steps. Finally, it sets $E_{i+1} \leftarrow E_i \backslash \{(v_i, v_j)\}$, and terminates if $E_{i+1} = \emptyset$.

We note that the idea in the above batch-ordering scheme is to establish an ordering using the strongest edges in G. This will be an effective defense against the Condorcet attack if the ordering of the honest transactions has "strong support" in the system. In a special case where all the nodes are honest, and all support/report the same ordering of honest transactions, the Condorcet attack can be fully prevented as stated in the following theorem.

▶ **Proposition 4.** *Suppose that the Condorcet attack succeeds in creating a Condorcet cycle. Let $\texttt{tx}_1, \texttt{tx}_2, \ldots, \texttt{tx}_m$ be the set of honest transactions in the Condorcet cycle. Suppose that all the nodes in the system are honest and report $\texttt{tx}_i$ before $\texttt{tx}_j$ for every $1 \leq i < j \leq m$. Then the proposed ranked pairs batch-ordering scheme returns the true ordering of the honest transaction, that is it orders $\texttt{tx}_i$ before $\texttt{tx}_j$ for every $1 \leq i < j \leq m$.*

[3] besides Schulze, ranked pairs is the only existing electoral system that satisfies anonymity, Condorcet criterion, resolvability, Pareto optimality, reversal symmetry, monotonicity, and independence of clones [18].

[4] When there are multiple edges with the highest weight, one can be chosen according to a fixed tie-breaking method.

Proof. Let $G = (V, E)$ be the graph with V representing the transactions in the Condorcet cycle, and the weight of each edge $(u, v) \in E$, represented as $w(u, v)$, be equal to the number of nodes that reported u before v. Let $u_1, u_2, \ldots, u_m$ be the vertices in V that represent the honest transactions. Let $E_f \subseteq E$ be the set of all edges with the full support of the nodes, that is

$$E_f = \{e \in E | w(e) = n\},$$

where n is the number of nods in the system. Since all the nodes in the system have the same view on the ordering of the honest transactions, we get that $(u_i, u_j) \in E_f$ for every $1 \leq i < j \leq m$. We note that the sub-graph $G' = (V, E_f)$ of G is cycle free, as otherwise there will be a cycle in the ordering of individual nodes. The ranked pairs batch-ordering algorithm first chooses all the edges in E_f before proceeding with other edges in E. When the algorithm covers all the edges in E_f the true ordering of the honest transactions will be set, and cannot be changed by the remaining steps of the algorithm. ◀

Limitation. Proposition 4 considers an ideal scenario where 1) all the nodes are honest, and 2) they all report the honest transaction in the same order. If one of the above two conditions does not hold, however, the Condorcet attack may be able to create a cycle (see the following example).

▶ **Example 5.** Consider a system with $n = 5$ nodes. Let $\mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3$ be three honest transactions. An adversarial client $\mathcal{C}$ can create a Condorcet cycle of the form

$$\begin{aligned}
N_1 &: \quad [\mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_3, \mathtt{A}_4, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3] \\
N_2 &: \quad [\mathtt{A}_2, \mathtt{A}_3, \mathtt{A}_4, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}_1] \\
N_3 &: \quad [\mathtt{A}_3, \mathtt{A}_4, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}_1, \mathtt{A}_2] \\
N_4 &: \quad [\mathtt{A}_4, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_3] \\
N_5 &: \quad [\mathtt{tx}_3, \mathtt{tx}_2, \mathtt{tx}_1, \mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_3, \mathtt{A}_4]
\end{aligned}$$

where $\mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_3, \mathtt{A}_4$ are the transactions submitted by $\mathcal{C}$. Note that all the nodes, except Node 5, report the order $[\mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3]$, while node 5 reports $[\mathtt{tx}_3, \mathtt{tx}_2, \mathtt{tx}_1]$ (Node 5 is either controlled by the adversary or is an honest node who has simply received the transactions in this order). If we run the proposed ranked pairs batch-ordering scheme on this cycle, the returned order of honest transactions may be incorrect. It is because the edge between any pair of transactions has a weight of 4 in the dependency graph. As a result, an edge between two honest transactions such as $\mathtt{tx}_1$ and $\mathtt{tx}_2$ may be eliminated in the ranked pairs method, which would result in $\mathtt{tx}_2$ and $\mathtt{tx}_3$ to be ordered before $\mathtt{tx}_1$. As for Themis, if we use the proposed method by Yannis Manoussakis [14] (as suggested by Themis), we get the Hamiltonian cycle $(\mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_3, \mathtt{A}_4, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3)$. All the edges in this cycle have the identical weight of four, hence there is no distinct weakest edge. Therefore, Themis may remove any of the edges in the cycle. If the removed edge is between two honest transactions, the final ordering of honest transactions would be incorrect. We remark that both the ranked pairs batch-ordering scheme and Themis would order honest transactions correctly if Node 5 order honest transactions as $[\mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3]$. This example, therefore, shows that the adversary has more power in modifying the order of honest transactions if (in addition to injecting transactions) it controls a number of nodes in the system (e.g. Node 5 in this example).

▶ Remark 6. To use the proposed ranked pairs batch-ordering scheme in Themis, we can simply replace the Hamiltonian-based batch-ordering scheme of Themis with the ranked pairs batch-ordering scheme in the FairFinalize algorithm. We remark that the weight information of the dependency graph is available within the FairFinalize algorithm, thus this replacement is possible.

6.2 Post-decryption Resolution

In secure causal ordering, as mentioned earlier, transactions are ordered while they are encrypted, and get decrypted only once a total ordering is committed [17, 3]. This prevents an adversary from observing the contents of transactions while they are being ordered, hence eliminating those front-running attacks (e.g. the sandwich attack [16]) that must examine the content of transactions.

To mitigate the Condorcet attack, we propose to maintain the above strategy, except we leave the ordering of transactions inside a Condorcet cycle to after they are decrypted. Note that after the decryption of these transactions, an adversary cannot impose a change to the ordering as 1) there is already a consensus on the set of transactions that must be included, thus the adversary cannot add or remove any transaction to the set; 2) the ordering of the transactions is performed locally at each node using a pre-determined algorithm. In other words, it is too late for the adversary to manipulate the ordering of transactions, although the contents of transactions are disclosed.

Once the transactions within a cycle are decrypted, their contents are disclosed, enabling them to be partitioned into independent groups (i.e., transactions inside different groups are independent of each other). Each group can then be ordered independent of the others. By implementing this measure, the adversary is unable to manipulate the ordering of honest transactions if the adversary's transactions are independent of honest transactions. This is because the adversary's transactions will not fall within any group that includes honest transactions. Note that we still need to order the groups themselves (i.e. which group comes first, which comes second, and so on). As transactions across various groups have no effect on one another, the groups can be safely ordered using a pre-determined algorithm such as ranked pairs as described in Section 6.1.

▶ Remark 7. In the Themis protocol, we can apply the above post-decryption resolution method within the FairFinalize algorithm: If transactions `A` and `B` are independent, the edge between them in the dependency graph can be safely removed.

Limitation. The post-decryption resolution prevents the adversary from manipulating the order of honest transactions if the adversary's transactions are independent of the honest transactions. In certain scenarios, however, the adversary may be able to create dependencies. For instance, consider a situation where a popular NFT is dropping in a block currently being formed. Given the high demand, many transactions are transmitted with the intention of acquiring this NFT. Recognizing this, the adversary can execute the Condorcet attack by using transactions that fall within the same dependency group as those attempting to acquire the NFT.

Another limitation of the post-decryption resolution is the computational burden it places on the system to identify dependencies between transactions.

6.3 Broadcast

In the Condorcet attack, the adversary follows a well-structured three-phase strategy: in the first phase, the adversary sends a set of transactions, then pauses in the second phase, and then finishes the attack by sending another round of transactions in the third phase. The idea behind our third mitigation technique is to disturb/break the above pattern by broadcasting transactions inside the system as soon as they arrive at an honest node. Because of the broadcast, the adversary's transactions that were submitted in the first phase will propagate in the system, which can nullify the adversary's target in the third phase since the transactions that the adversary transmits in the third phase have already been received by the nodes (thus their order has already been decided by the nodes).

In Section 7.5, we observe that this strategy proves highly effective in mitigating the suggested Condorcet attack. However, it is important to note that this strategy does incur increased communication overhead as a drawback. For instance, in Themis, nodes transmit transactions only to the leader as opposed to broadcasting in the network by themselves. Therefore, when applied to Themis, the above strategy will increase Themis's communication overhead (although it does not increase Themis's quadratic communication complexity).

Limitation. The main limitation of the above broadcast-based mitigation technique is that it will be ineffective if the adversary has strong control over the internal network. For instance, in Themis and Aequitas, it is assumed that the adversary controls all message delivery in the internal network, and can delay messages up to a bound Δ. If Δ is large enough (e.g., if it is larger than the duration of the Condorcet attack) then the adversary can circumvent the proposed mitigation by delaying all the broadcast transactions so they are delivered only after the attack is complete.

7 Simulation

To assess the impact of the Condorcet attack, as well as the effectiveness of the proposed mitigation methods, we conduct a series of experiments through simulations. In this section, we present the results of these experiments.

Environments. Our simulation encompasses four environments. The first environment captures the honest setting, where all the nodes and clients are honest, thereby eliminating the possibility of a Condorcet attack. Even in this environment, Condorcet cycles can occur. Therefore, we are interested to know if our proposed ranked pairs batch-order scheme can more effectively order transactions within a cycle than the Hamiltonian-cycle-based scheme used in Themis.

In the second environment, all the nodes in the system are honest, but there is an external adversary, who conducts the Condorcet attack from outside the system. In this environment, we are interested to evaluate the success rate and impact of the Condorcet attack (i.e., how many honest transactions the adversary can trap within a cycle).

In the third environment, we introduce packet reordering to the external network. We evaluate the impact of this on the success rate of the Condorcet attack. We also observe how the cloning method can help the adversary to improve its success rate.

The last environment that we consider is similar to the second environment, except this time we guard the system using the proposed mitigation methods. In this environment, we measure the impact of the Condorcet attack in order to examine the strength of the proposed mitigation methods.

Clients. We use a sending process to submit all the clients' transactions to the system. The sending process transmits transactions in sequence at discrete times t_i, $i \geq 0$. At each time instance, the process sends (n copies of) the transaction of a given client to all the n nodes in the system. Each copy of the transaction will arrive at its destination node with a random delay drawn independently from a distribution named `NetworkDist`. We refer to this distribution as the network latency. We use another distribution, `GenerationDist`, to determine the delay between two consecutive time instances (i.e. $t_{i+1} - t_i$ follows the `GenerationDist` distribution). Similar to [11], we set both `GenerationDist` and `NetworkDist` to exponential distributions with means of one and r, respectively. We refer to r as *external network ratio*. One can think of r as the expected number of clients who transmit transactions within a time frame equal to the average network latency.

Themis Variant. In our simulations, we use the practical Themis variant with the communication complexity of $\mathcal{O}(n^2)$, instead of the the SNARK-Themis variant. In our simulations, all transactions are eventually received by each node in a single round. Therefore, the choice of γ does not have any impact on the simulation results (hence, we simply set $\gamma = 1$). We used the latest version of Themis, which breaks the Hamiltonian cycle by removing the weakest link. The weakest link is the link that has the least weight or support in the Hamiltonian cycle. To construct a Hamiltonian cycle, we used the proposed method by Yannis Manoussakis [14] as suggested by Themis.

7.1 Honest Environment

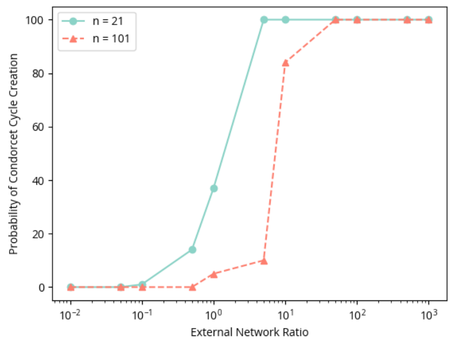
Honest Environment Setting. In this environment, all the nodes and clients are honest, and consequently, there is no Condorcet attack. Nevertheless, as shown in Figure 2, Condorcet cycles can occur particularly when the external network ratio is greater than one.

To obtain the results plotted in Figure 2, we varied the external network ratio from 0.01 to 1000. For each given network ratio, and each network size of $n = 21$ and $n = 101$, we conducted 100 simulation runs. In each run, the sending process transmitted 100 transactions (at 100-time instances drawn from the `GenerationDist` distribution). Once every node received all the transmitted transactions, we proceeded to generate the dependency graph using the Themis algorithm. By examining the graph (i.e. extracting strongly connected components) we then identified all the Condorcet cycles.

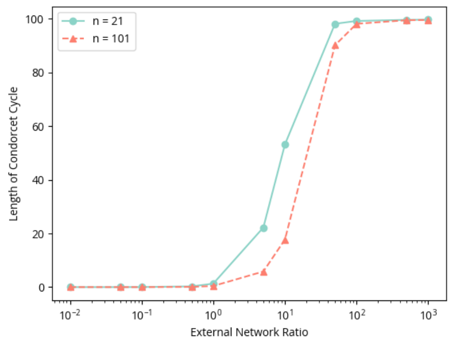
Cycle Length. An interesting observation from Figure 2 is that when the external network ratio is less than about one, Condorcet cycles rarely occur. As the external network ratio becomes larger than one, however, Condorcet cycles start to appear. For high values of the external network ratio, as depicted in Figure 2, Condorcet cycles not only occur frequently but also include many of the transmitted transactions. Overall, this observation suggests a critical threshold at which the system's behavior, with respect to creating Condorcet cycles, significantly changes.

Condorcet Cycles Categories. We refer to Condorcet cycles that are not created by an adversary as *natural Condorcet cycles*. Conversely, we call a Condorcet cycle adversarial if it is created by an adversary. In Section 6.1, we proposed a ranked pairs batch-ordering scheme to handle the ordering of transactions within an adversarial Condorcet cycle. Later in this section, we demonstrate that the proposed scheme indeed alleviates the severity of the Condorcet attack.

Ranked Pairs Performance. Here, we show (Figure 3) that the proposed ranked pairs batch-ordering scheme is also a good candidate for ordering transactions within a natural Condorcet cycle. Consequently, even in an honest environment, we can improve fairness in ordering transactions by replacing the existing batch-ordering schemes (i.e., the alphabetical scheme, and the Hamiltonian-based scheme of Themis) with the proposed ranked pairs batch-ordering scheme.



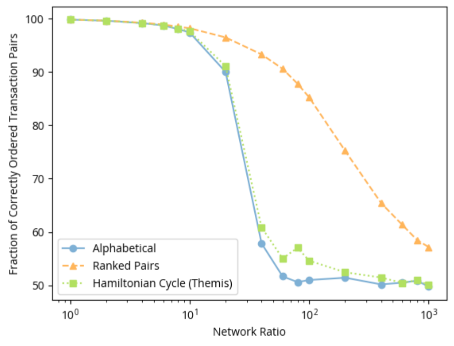
(a) The chance of a Condorcet cycle.



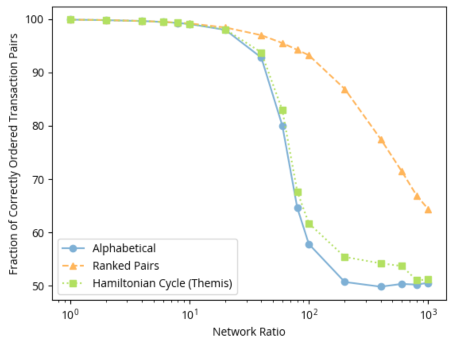
(b) Number of transactions in cycles.

Figure 2 Condorcet cycles in the honest environment.

Batch Ordering-Schemes Performance Comparison. In Figure 3, the external network ratio (the x-axis) ranges from 1 to 1000; this is the range in which Condorcet cycles naturally occur. The y-axis shows the fraction of transaction pairs that are ordered correctly according to their transmission time. Each data point in Figure 3 is the average of values obtained over 100 simulation runs. The data presented in this figure demonstrate the superiority of the proposed ranked pairs batch-ordering scheme for two network sizes of $n = 21$ and $n = 101$.



(a) The number of nodes is $n = 21$.



(b) The number of nodes is $n = 101$.

Figure 3 Fraction of correctly ordered transactions in the honest environment.

7.2 Adversarial Environment

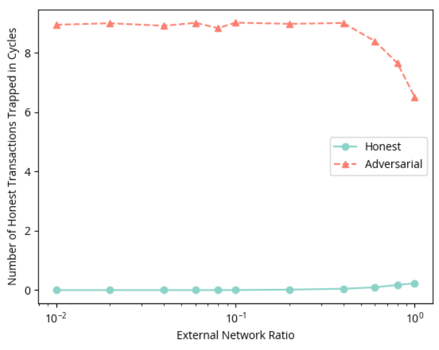
Adversarial Environment Setting. In the existing adversarial environments in the literature, there is often at least one (typically up to $f = \theta(n)$) adversarial node in the system. In our adversarial environment, in contrast, all the nodes in the system can be honest. There is, however, an adversarial client in our environment who orchestrates the Condorcet attack from outside the system.

In this section, we evaluate the performance of the Condorcet attack in this environment. In particular, we measure the success rate of the attack in the number of honest transactions it can trap within a cycle. The measurement is carried out for external network ratios r less than one, as natural Condorcet cycles are rare in this regime, particularly when $r \ll 1$. This allows us to assess the strength of the attack in creating cycles in a setting where Condorcet cycles do not naturally happen.

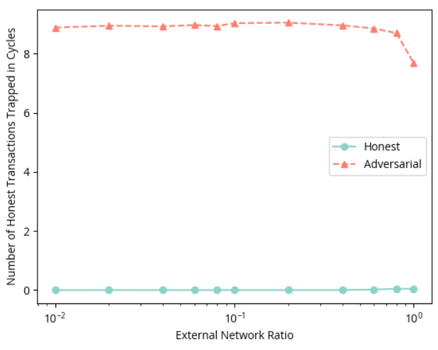
In our simulation, we simply use two adversarial transactions to create the Condorcet cycle as described in Example 1. We set the pause time of the Condorcet attack to $\tau \in \{10, 50\}$ times the mean of the `GenerationDist` distribution. This means that, on average, τ honest transactions are transmitted to the system during the pause time.

In parallel to the transmissions of honest transactions, the two adversarial transactions are transmitted to create a Condorcet cycle. Once all transactions are received by the nodes, we calculate two separate dependency graphs: one considering the adversarial transactions, and one ignoring them. By comparing these two dependency graphs, we then assess the impact of the attack on the final ordering.

Condorcet Attack Performance. Figures 4 and 5 show the average number of the honest transactions that the attack can trap within cycles over two different settings: $\tau = 10$ and $\tau = 50$. As shown, for a wide range of external network ratios, the attack can trap nearly all the honest transactions that are transmitted during the pause time (about 9 honest transactions in the setting $\tau = 10$, and nearly 49 honest transactions in the setting $\tau = 50$). This demonstrates the strength of the attack, considering that, on average τ honest transactions are submitted to the system during the pause time (and the attack traps nearly all of them).



(a) $\tau = 10$, $n = 21$.

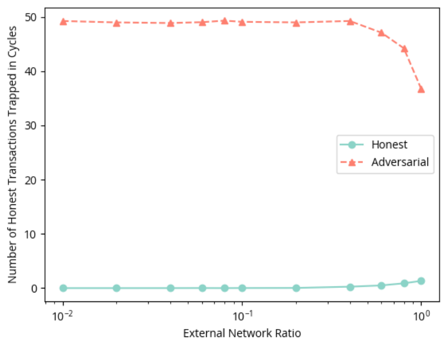


(b) $\tau = 10$, $n = 101$.

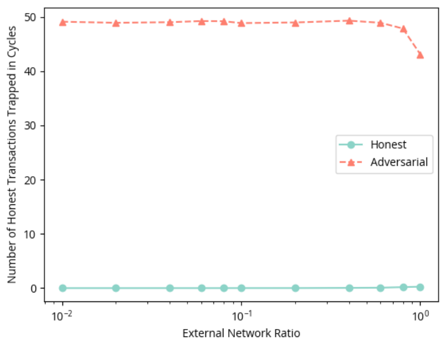
Figure 4 Number of honest transactions trapped in Condorcet cycles for $\tau = 10$.

7.3 Network Reordering

In the Condorcet attack, the adversary sends a sequence of transactions in a particular order to create a cycle. The external network may, however, change the order of transactions transmitted, which can, in turn, reduce the attack's success rate. To evaluate this, we performed simulations over a network which changes the order of two consecutively transmitted transactions with probability $0 \leq p \leq 0.5$. For each value of p, we performed 1000 runs of simulations. The success rate of the attack was set to the fraction of runs in which the attack successfully trapped the honest transactions in a Condorcet cycle.



(a) $\tau = 50$, $n = 21$.



(b) $\tau = 50$, $n = 101$.

Figure 5 Number of honest transactions trapped in Condorcet cycles for $\tau = 50$.

Using the above setting, we conducted two instances of the Condorcet attack. The first instance uses two adversarial transactions A and B as in Example 1, and takes the following pattern:

$$\begin{aligned} P_1 &: \quad \texttt{A}, \texttt{B}, \text{Pause} \\ P_2 &: \quad \texttt{B}, \text{Pause}, \texttt{A} \\ P_3 &: \quad \text{Pause}, \texttt{A}, \texttt{B} \end{aligned}$$

As illustrated in Figure 6, this instance is sensitive to network reordering (the success rate of the attack drops quickly with p). As shown in the figure, the attack's success rate increases when we use the second instance of cloning described below.

In our second instance (denote as $\texttt{tx} = 4$ in Figure 6), the adversary partitions nodes into four parts P_1, P_2, P_3 and P_4, and uses four transactions (A, B, C and D) instead of two, in the following pattern:

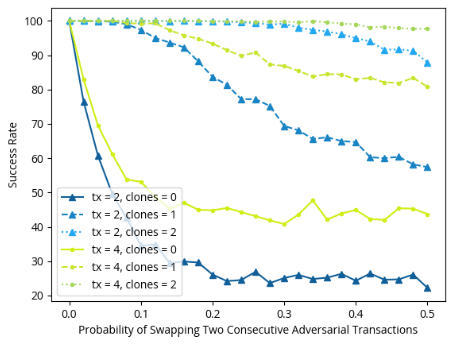
$$\begin{aligned} P_1 &: \quad \texttt{A}, \texttt{B}, \text{ Pause }, \texttt{C}, \texttt{D} \\ P_2 &: \quad \texttt{B}, \texttt{C}, \text{ Pause }, \texttt{D}, \texttt{A} \\ P_3 &: \quad \texttt{C}, \texttt{D}, \text{ Pause }, \texttt{A}, \texttt{B} \\ P_4 &: \quad \texttt{D}, \texttt{A}, \text{ Pause }, \texttt{B}, \texttt{C} \end{aligned}$$

This instance of the Condorcet attack is more robust against network reordering as demonstrated in Figure 6. As in the first instance, the success rate of the instance can be boosted using the cloning method. In particular, note that the second instance together with a single clone is almost fully resistant to network transaction reordering.

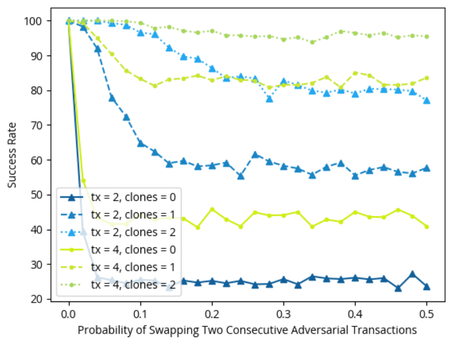
7.4 A Non-Injective Condorcet Attack

Injecting transactions into the system is a key component of the proposed Condorcet attack. Without this component, an adversary has limited power in creating cycles even when the adversary controls the leader and a faction of all the nodes in the system.

To illustrate the above point, we conducted simulations over two networks with sizes: $n = 21$ and $n = 101$. In our simulation, the adversary controls the maximum fraction of nodes, including the leader, allowed by Themis (a quarter of nodes minus one). All these



(a) The number of nodes is $n = 21$.



(b) The number of nodes is $n = 101$.

Figure 6 Impact of network reordering on the success of the Condorcet attack.

nodes report the order of their received transactions in reverse, in a strategy to create Condorcet cycles[5]. The external network ratio is varied from 0.01 to 100 to capture a wide range of network conditions. The total number of transmitted transactions is set to 100.

To evaluate the impact of the above strategy in creating cycles, we created two dependency graphs. The first graph represents the scenario where the adversarial nodes reverse their orderings, whereas the second graph represents the scenario where the adversarial nodes report the true ordering. Figure 7 shows the results of our simulation.

Non-Injective Condorcet Attack Performance. As shown in Figure 7, the adversary's attempts to create cycles are largely unsuccessful in the region where the external network ratio is less than one. We note that in this region, the average temporal gap between two different transaction transmissions is more than the average network latency. In particular, when $r \ll 1$ (i.e., when transactions are transmitted far apart in time with respect to the network latency), honest nodes in the system have a clear view of the true ordering of transactions. In this region, the adversary is all but powerless in creating cycles[6], as evident in Figure 7. In contrast, in the same region, an external adversary can create a cycle using the proposed Condorcet attack, even when all the nodes in the system are honest.

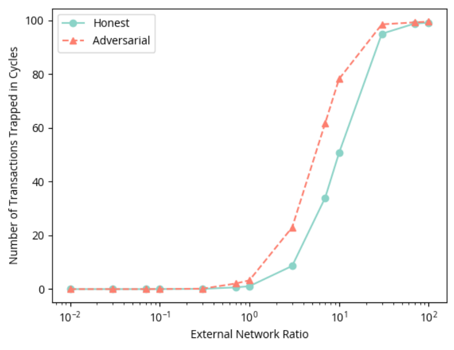
7.5 Mitigation

In this section, we evaluate the performance of our mitigation methods in preventing or minimizing the impact of the Condorcet attack.

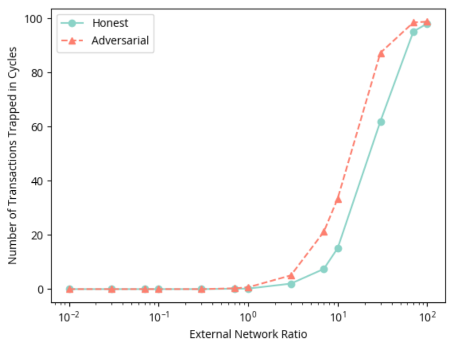
Ranked-pairs-based Mitigation Method. To evaluate the effectiveness of this mitigation, we conducted a simulation over two network sizes of $n = 21$ and $n = 101$. We set the pause time of the attack to 10 times the mean of `GenerationDist`, and set the total number of honest transactions to 20. We varied the external network ratio r from 0.001 to 1. Recall

[5] We note that this may not be an optimum strategy to create Condorcet cycles. Nevertheless, we believe that an optimum strategy (which may be computationally intractable) may not be significantly more successful than the adopted strategy. We leave the validation of this claim for future work.

[6] When $r > 1$ (i.e., in the region where Condorcet cycles naturally emerge) the adversary achieves some degree of success in creating larger cycles than naturally occur.



(a) The number of nodes is $n = 21$.

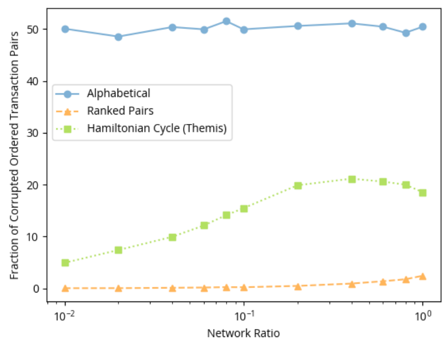


(b) The number of nodes is $n = 101$.

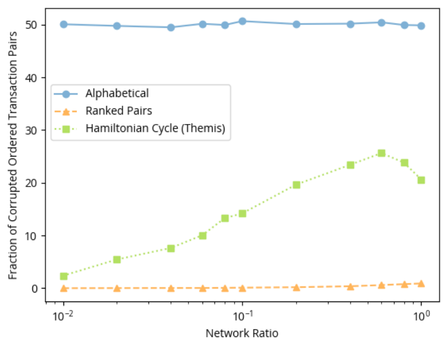
Figure 7 The non-injective attack has limited power in creating cycles.

that in this range of external network ratio (i.e., $r < 1$), Condorcet cycles do not emerge naturally; rather they are created by the Condorcet attack. To evaluate the true impact of our ranked-pars mitigation method, therefore, we focused on this region.

Ranked Pairs Mitigation Performance. Figure 8 compares the performance of our proposed ranked-pairs-based mitigation method to the Hamiltonian-based method used in Themis, and the simple alphabetical method. The results show that the proposed ranked-pairs method achieves a low error rate, indicating that it can effectively order honest transactions correctly even when they fall in a Condorcet cycle. In contrast, the Themis algorithm's error rate increases as the network ratio increases, and reaches as high as about 25%. The error rate in the case of alphabetical ordering is 50%. Note that a random ordering method can, on average, correctly orders 50% of all the pairs of transactions. In this sense, the worst-case transaction ordering error is 50%, which is the case for the alphabetical method (this method is essentially a random ordering method).



(a) The number of nodes is $n = 21$.



(b) The number of nodes is $n = 101$.

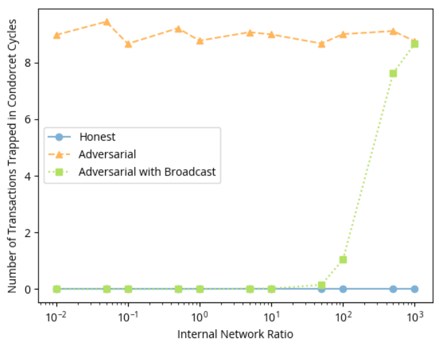
Figure 8 The performance of the proposed ranked pairs-based mitigation method.

The Broadcast-based Mitigation Method. To evaluate the effectiveness of the broadcast-based mitigation method, we conducted simulations using two network sizes: $n = 21$ and $n = 101$. We introduced a new exponential distribution called `InternalNetworkDist`, which captures the random delays experienced by messages within the internal network. Specifically, we sample from `InternalNetworkDist` to determine the delay between sending a transaction from one node to another node. This is in contrast to `NetworkDist`, which is used to determine the random delays between a client and a node in the external network.

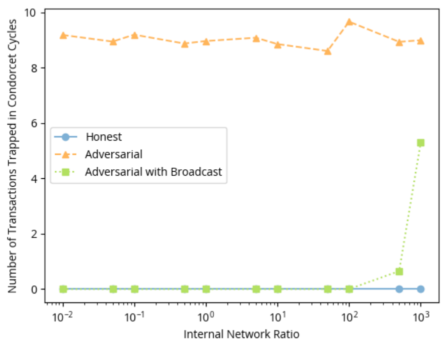
In our simulation, we set the mean of `InternalNetworkDist` to r'. We refer to r' as the *internal network ratio*. In our simulations, we set τ to 10 times the mean of `GenerationDist` (i.e. $\tau = 10 \cdot r$), and set the total number of honest transactions to 20. We fixed the external network ratio to $r = 0.1$, to ensure that no natural Condorcet cycles were created, and varied the internal network ratio r' from 0.01 to 1000.

Broadcast Environment Categories. We analyzed the number of honest transactions trapped in a Condorcet cycle under three different settings. In the first setting, referred to as the "honest setting", nodes did not broadcast and the adversary did not conduct a Condorcet attack. In the second setting, nodes still did not broadcast, but the adversary attempted a Condorcet attack. Finally, in the last setting, the adversary launched an attack while the nodes employed the broadcasting method to mitigate it.

Broadcast Mitigation Performance. Figure 9 shows the result of our simulations in the above three settings. The results demonstrate that the proposed broadcast-based mitigation is highly effective in preventing the adversary from creating a Condorcet cycle and trapping honest transactions. This can be attributed to two key factors: Firstly, the mitigation strategy disrupts the completion of the pause phase, thereby preventing honest transactions from being trapped in a Condorcet cycle. When the internal network ratio r' is smaller than the pause time, almost no transactions are trapped. Interestingly, even when r' exceeds the pause time, the adversary cannot achieve the same level of performance. It is because the broadcast of transactions with the internal network can still somewhat disturb the ordering of adversarial transactions. This reduces the success rate of the attack as the specific ordering of adversarial transactions is crucial for creating a Condorcet cycle. If, on the other hand, the adversary has enough control over the internal network to delay transactions as much as the pause time, it can circumvent the proposed broadcast-based mitigation as the adversary can enforce the ordering of its transactions within the internal network by delaying all the messages.



(a) The number of nodes is $n = 21$.



(b) The number of nodes is $n = 101$.

Figure 9 The performance of the proposed broadcast mitigation method.

8 Conclusion

Condorcet cycles can occur naturally. While these natural cycles may not significantly disrupt fairness in the system since transactions falling within these cycles are typically received around the same time, the artificial creation of Condorcet cycles can lead to significant unfairness in the system. In this paper, we showed that even with all nodes in the system behaving honestly, it is relatively simple to generate such artificial cycles. Furthermore, we demonstrated that these created cycles possess significant power, as they can trap transactions submitted at widely different times that would not naturally fall within a cycle.

To address this attack, we proposed three mitigation methods using different approaches. These methods complement one another and can be employed collectively to fortify the defensive measures against the attack. Through simulations, we showcased that despite their described limitations, the proposed mitigation methods can substantially reduce the adverse impact of the Condorcet attack.

References

1 Carsten Baum, James Hsin-yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. Sok: Mitigation of front-running in decentralized finance. *IACR Cryptol. ePrint Arch.*, page 1628, 2021. URL: `https://eprint.iacr.org/2021/1628`.

2 Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors. *Handbook of Computational Social Choice*. Cambridge University Press, 2016. `doi:10.1017/CBO9781107446984`.

3 Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2001. `doi:10.1007/3-540-44647-8_31`.

4 Christian Cachin, Jovana Micic, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness. In *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, Lecture Notes in Computer Science, pages 316–333. Springer, 2022. `doi:10.1007/978-3-031-18283-9_15`.

5 M. d. Condorcet. Essay on the application of analysis to the probability of majority decisions. *Paris: Imprimerie Royale*, 1785.

6 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *CoRR*, abs/1904.05234, 2019. `arXiv:1904.05234`.

7 Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. `doi:10.1145/42282.42283`.

8 Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: Front-running attacks on blockchain. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *Financial Cryptography and Data Security - FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11599 of *Lecture Notes in Computer Science*, pages 170–189. Springer, 2019. `doi:10.1007/978-3-030-43725-1_13`.

9 Lioba Heimbach and Roger Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. *CoRR*, abs/2203.11520, 2022. `doi:10.48550/arXiv.2203.11520`.

10 Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. In Jason Paul Cruz and Naoto Yanai, editors, *APKC '22: Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop, APKC@AsiaCCS 2022, Nagasaki, Japan, 30 May 2022*, pages 3–14. ACM, 2022. `doi:10.1145/3494105.3526239`.

11 Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *IACR Cryptol. ePrint Arch.*, page 1465, 2021. URL: `https://eprint.iacr.org/2021/1465`.

12 Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 451–480. Springer, 2020. `doi:10.1007/978-3-030-56877-1_16`.

13 Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 25–36. ACM, 2020. `doi:10.1145/3419614.3423263`.

14 Yannis Manoussakis. A linear-time algorithm for finding hamiltonian cycles in tournaments. *Discret. Appl. Math.*, 36(2):199–201, 1992. `doi:10.1016/0166-218X(92)90233-Z`.

15 David C. Parkes and Lirong Xia. A complexity-of-strategic-behavior comparison between schulze's rule and ranked pairs. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5075`.

16 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 198–214. IEEE, 2022. `doi:10.1109/SP46214.2022.9833734`.

17 Michael K. Reiter and Kenneth P. Birman. How to securely replicate services. *ACM Transactions on Programming Languages and Systems*, 16(3):986–1009, 1994. `doi:10.1145/177492.177745`.

18 Markus Schulze. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Soc. Choice Welf.*, 36(2):267–303, 2011. `doi:10.1007/s00355-010-0475-4`.

19 T. N. Tideman. Independence of clones as a criterion for voting rules. *Social Choice and Welfare*, 4(3):185–206, September 1987. `doi:10.1007/bf00433944`.

20 Yunhao Zhang, Srinath T. V. Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 633–649. USENIX Association, 2020. URL: `https://www.usenix.org/conference/osdi20/presentation/zhang-yunhao`.

Pay Less for Your Privacy: Towards Cost-Effective On-Chain Mixers

Zhipeng Wang
Imperial College London, UK

Marko Cirkovic
University of Bern, Switzerland

Duc V. Le[1]
Visa Research, Sunnyvale, CA, USA

William Knottenbelt
Imperial College London, UK

Christian Cachin
University of Bern, Switzerland

Abstract

On-chain mixers, such as Tornado Cash (TC), have become a popular privacy solution for many non-privacy-preserving blockchain users. These mixers enable users to deposit a fixed amount of coins and withdraw them to another address, while effectively reducing the linkability between these addresses and securely obscuring their transaction history. However, the high cost of interacting with existing on-chain mixer smart contracts prohibits standard users from using the mixer, mainly due to the use of computationally expensive cryptographic primitives. For instance, the deposit cost of TC on Ethereum is approximately $1.1M$ gas (i.e., 66 `USD` in June 2023), which is $53\times$ higher than issuing a base transfer transaction.

In this work, we introduce the Merkle Pyramid Builder approach, to incrementally build the Merkle tree in an on-chain mixer and update the tree per batch of deposits, which can therefore decrease the overall cost of using the mixer. Our evaluation results highlight the effectiveness of this approach, showcasing a significant reduction of up to $7\times$ in the amortized cost of depositing compared to state-of-the-art on-chain mixers. Importantly, these improvements are achieved without compromising users' privacy. Furthermore, we propose the utilization of verifiable computations to shift the responsibility of Merkle tree updates from on-chain smart contracts to off-chain clients, which can further reduce deposit costs. Additionally, our analysis demonstrates that our designs ensure fairness by distributing Merkle tree update costs among clients over time.

2012 ACM Subject Classification Security and privacy → Pseudonymity, anonymity and untraceability

Keywords and phrases Privacy, Blockchain, Mixers, Merkle Tree

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.16

Related Version *Full Version*: `https://eprint.iacr.org/2023/1222` [39]

Funding *Duc V. Le*: Supported by a grant from Protocol Labs to the University of Bern.

Acknowledgements The authors thank anonymous reviewers for helpful feedback.

[1] The main part of the work was conducted while the author was at the University of Bern.

1 Introduction

Permissionless blockchains, such as Bitcoin [29] and Ethereum [41], provide pseudonymity rather than complete anonymity. Each transaction is openly recorded in plaintext on the public ledger, revealing details such as the transaction amount, timestamp, and the addresses of the sender and recipient. While this information alone may not directly expose the individuals involved, it poses a risk as malicious actors can exploit it to analyze and link addresses, potentially compromising the anonymity of users [15, 44, 37, 38].

To enhance the privacy on non-privacy-preserving blockchains, on-chain mixers [4, 6, 1, 23] have been proposed. On-chain mixers are Decentralized Applications running on a blockchain with smart contracts, in which users deposit a *fixed* amount of coins into a pool, and subsequently withdraw those coins to a different address. When being used properly, on-chain mixers enable *unlinkability* between the deposit and the withdrawal addresses. The most active on-chain mixer, Tornado Cash (TC), has accumulated more than $51K$ unique deposit addresses for its largest pool [38]. The set of deposit addresses is similar to k-anonymity, which allows a withdrawal to be hidden among a set of k other deposits. The larger anonymity set size is accumulated, the harder it can link a user's withdrawal address to the corresponding deposit address.

However, achieving unlinkability through on-chain mixers comes at the cost of expensive deposit operations, making it difficult for regular users to use them. In particular, on-chain mixers use Merkle tree with Snark-friendly hash functions such as MiMC [7] and Poseidon [18] to record deposit and withdrawal history. When a deposit happens, the new leaf is appended sequentially to the Merkle tree. This update operation is the most expensive operation for on-chain mixers. In particular, depositing into TC costs approximately $1.1M$ gas, equating to roughly 66 USD as of June 2023. Such high costs can potentially discourage users from utilizing the mixer, thereby impeding the growth rate of the anonymity set size. Consequently, it becomes crucial to address these expensive deposit costs to ensure wider adoption and encourage the continued expansion of the anonymity set size.

In this paper, we propose two approaches to reduce the overall deposit costs in on-chain mixers. Firstly, we introduce the Merkle Pyramid Builder approach, in which we batch the deposits together and renew the Merkle tree per batch. This approach provides a promising solution to the scalability and efficiency challenges of on-chain mixers. Secondly, we propose the utilization of off-chain verifiable computation to further minimize deposit costs. This approach empowers clients to perform Merkle tree updating computations locally and provide cryptographic proofs to validate the accuracy of these computations. By shifting the responsibility of updating the Merkle tree from on-chain smart contracts to off-chain clients, we can significantly reduce the associated costs, enhancing the overall system efficiency.

Our contributions can be summarized as follows.

1. Merkle Pyramid Builder. We propose the Merkle Pyramid Builder (MPB) approach, which batches the deposits in a queue and reduces the Merkle tree update times in an on-chain mixer. This approach decreases the average number of times to execute the expensive Merkle tree with smart contracts per deposit. We locally implement the improved mixer with MPB construction, and our evaluation results demonstrate its remarkable ability to reduce deposit costs by 7× compared to the widely adopted TC mixer.

2. Off-Chain Deposit Proof Generation. We employ Verifiable Computation (VC) techniques to further minimize the cost of updating the Merkle tree. This is accomplished by offloading computations to the off-chain environment, effectively eliminating the computational requirements imposed on the smart contract. Our empirical evaluation results indicate that this approach further reduces the cost of a single deposit update.

3. Formal On-Chain Mixer Analysis Framework. We provide a formal framework to analyze the on-chain mixers by considering the properties of *correctness*, *privacy*, *availability*, *efficiency*, and *fairness*. We prove that our deposit-cost-reduction approaches offer enhanced *efficiency* without deteriorating other properties. Particularly, our analysis shows that the improved mixers can also guarantee *fairness*, with which the clients' costs for interacting with the mixer can be amortized over time. Furthermore, our analysis framework is of independent interest and could be applied to analyze other mixer designs.

The full version of this paper is available at [39].

2 Preliminaries

2.1 On-chain Mixers

On non-privacy-preserving blockchains, transactions are recorded in plaintext on the public ledger. On-chain mixers, inspired by Zerocash [32], are one of the most widely-used privacy solutions for non-privacy-preserving blockchains. On-chain mixers are running on top of blockchains with smart contracts, e.g., Ethereum and Binance Smart Chain (BSC). Upon using a mixer, a user deposits a fixed denomination of coins into a pool and later withdraws these coins to another address [4, 5, 6, 1, 23]. When used properly, mixers can break the linkability between addresses, and thus enhance users' privacy. The largest on-chain mixer, TC, has accumulated over $3.54M$ ETH from more than $39K$ Ethereum addresses [38].

2.2 Cryptographic Primitives

Notation. We denote by 1^λ the security parameter and by $\mathsf{negl}(\lambda)$ a negligible function in λ. We express by $(\mathsf{pk}, \mathsf{sk})$ a pair of public and private keys. Moreover, we require that pk can always be efficiently and deterministically derived from sk, and denote $\textsc{extract}\mathrm{PK}(\mathsf{sk}) = \mathsf{pk}$ to be the deterministic function to derive pk from sk. We denote $\mathbb{Z}_{\geq a}$ as the set of integers that are greater or equal a, $\{a, a+1, \dots\}$. We let PPT denote probabilistic polynomial time. We denote $st[a, b, c \dots]$ as an instance of the statement st where $a, b, c \dots$ have fixed and public values. We use a shaded area i, j, k to denote the private inputs in the statement $st : \{(a, b, c; i, j, k) : f(a, b, c, x, y, z) = \textsc{true}\}$.

Collision Resistant Hash Functions. A family H of hash functions is collision-resistant, iff for all PPT $\mathcal{A}$, given $h \xleftarrow{\$} H$, the probability that $\mathcal{A}$ finds x, x', such that $h(x) = h(x')$ is negligible. We refer to the cryptographic hash function h as a fixed function, $h : \{0,1\}^* \rightarrow \{0,1\}^\lambda$. For the formal definitions of the hash function family, we refer readers to [31].

zk-SNARK. A zero-knowledge Succinct Non-interactive ARgument of Knowledge (zk-SNARK) is a "succinct" non-interactive zero-knowledge proof (NIZK) for arithmetic circuit satisfiability. The construction of zk-SNARK is based on a field $\mathbb{F}$ and an arithmetic circuit C. An arithmetic circuit satisfiability problem of a circuit $C : \mathbb{F}^n \times \mathbb{F}^h \rightarrow \mathbb{F}^l$ is captured by the statement $st_C : \{(x, \mathsf{wit}) \in \mathbb{F}^n \times \mathbb{F}^h : C(x, \mathsf{wit}) = 0^l\}$, with the language $\mathcal{L}_C = \{x \in \mathbb{F}^n \mid \exists\ \mathsf{wit} \in \mathbb{F}^h\ s.t.\ C(x, \mathsf{wit}) = 0^l\}$.

▶ **Definition 1** (zk-SNARK). *zk-SNARK for arithmetic circuit satisfiability is a triple of efficient algorithms* ($\textsc{Setup}$, $\textsc{Prove}$, $\textsc{Verify}$)*:*

- $(\mathsf{ek}, \mathsf{vk}) \leftarrow \textsc{Setup}(1^\lambda, C)$ *takes as input the security parameter and the arithmetic circuit* C*, outputs an evaluation key* ek*, and a verification key* vk*.*
- $\pi \leftarrow \textsc{Prove}(\mathsf{ek}, x, \mathsf{wit})$ *takes as input the evaluation key* ek *and* $(x, \mathsf{wit}) \in st_C$*, outputs a proof* π *for the statement* $x \in \mathcal{L}_C$*.*
- $0/1 \leftarrow \textsc{Verify}(\mathsf{vk}, \pi, x)$ *takes as input the verification key* vk*, the proof* π*, the statement* x*, outputs* 1 *if* π *is valid proof for the statement* $x \in \mathcal{L}_C$*.*

Commitment Scheme. A commitment scheme allows an entity to commit to a value while keeping it hidden, with the option of later revealing the value. A commitment scheme contains two rounds: committing and revealing. During the committing round, a client commits to selected values while concealing them from others. The client can choose to reveal the committed value during the revealing round, and another entity can verify its consistency.

▶ **Definition 2** (Commitment Scheme). *A commitment scheme includes two algorithms:*

- $\mathsf{cm} \leftarrow \textsc{Commit}(m, r)$ *accepts a message* m *and a secret randomness* r *as inputs and returns the commitment string* cm*.*
- $0/1 \leftarrow \textsc{Verify}(m, r, \mathsf{cm})$ *accepts a message* m*, a commitment* cm *and a decommitment value* r *as inputs, and returns* 1 *if the commitment is opened correctly and* 0 *otherwise.*

A secure commitment scheme satisfies two requirements: *(i) Binding:* Except for a negligible probability, no adversary can efficiently create cm, (m_1, r_1), and (m_2, r_2) such that $\textsc{Verify}(m_1, r_1, \mathsf{cm}) = \textsc{Verify}(m_2, r_2, \mathsf{cm}) = 1$ and $m_1 \neq m_2$. *(ii) Hiding:* Except for a negligible probability, cm does not reveal any information about the committed data.

Authenticated Data Structure. An Authenticated Data Structure (ADS) is a data structure that not only stores information but also provides a cryptographic proof of the integrity and authenticity of its contents. It allows for efficient verification of data integrity without requiring the entire data structure to be transmitted or stored alongside the proof. In this work, we adopt the Merkle tree as an ADS for set membership proof.

▶ **Definition 3** (Merkle Tree). *A Merkle tree leverages a collision-resistant hash function* h *to construct the data structure. The four algorithms work as follows:*

- $\mathsf{root} \leftarrow \textsc{Init}(1^\lambda, X)$ *takes the security parameter and a list* $X = (x_1, \ldots, x_n)$ *as inputs, constructs a tree that stores* $x_1, \ldots, x_n$ *in the leaves, and finally outputs a root,* root*.*
- $\mathsf{path}_i \leftarrow \textsc{Prove}(i, x, X)$ *takes an element* $x \in \{0,1\}^*, 1 \leq i \leq n$ *and a list* $X = (x_1, \ldots, x_n)$ *as inputs, and outputs the proof* path_i*, which can prove that* x *is in* X*. The proof generation time is proportional to* n*, while the proof size grows logarithmically with* n*.*
- $0/1 \leftarrow \textsc{Verify}(i, x_i, \mathsf{root}, \mathsf{path}_i)$ *takes an element,* $x_i \in \{0,1\}^*$*, an index* $1 \leq i \leq n, y \in \{0,1\}^\lambda$ *and a proof* π *as inputs, and outputs* 1 *if* π *is correctly verified and* 0 *otherwise.*
- $y' \leftarrow \textsc{Update}(i, x, X)$ *takes an element* $x \in \{0,1\}^*, 1 \leq i \leq n$ *and* X *as inputs, and outputs* $y' = \textsc{Init}(1^\lambda, X')$ *where* X' *is* X *but* $x_i \in X$ *is replaced by* x*.*

A Merkle tree should satisfy *correctness* and *security.* For the formal definitions of these properties, we refer to the cryptography introduction book of Boneh and Shoup [13].

Cost for Appending to Merkle tree. Let T be a Merkle hash tree of size n and assume that all internal nodes are stored. If a single element is appended, the computational cost of updating a Merkle hash tree is $O(\log n)$ if the internal tree nodes are stored. This can be done by traversing the right-most path of the tree and modifying at most $O(\log n)$ internal nodes of the tree.

Inefficiency of Multiple Appending Operations in Existing On-Chain Mixers. We observe that despite the theoretically efficient append operation, the utilization of a SNARK-friendly hash function still leads to a $\log n$ cost of appending that amounts to around $1M$ gas. For k deposits, this operation is repeated k times, raising the cost to $O(k \cdot \log n)$.

However, in a conventional mixer, to assimilate within a sufficiently large anonymity set, users often wait several days before withdrawing funds [38, 23]. Consequently, this suggests that the Merkle tree update operation could be processed in a batch rather than individually each time. This adjustment could potentially decrease the cost from $O(k\log(n))$ to $O(k + \log(n))$. Further details on this improvement will be provided in Section 5.

Efficient Replace. The update algorithm described previously needs the entire set X to be able to recalculate the root. Nevertheless, it is feasible to update the root without knowing the entire set. Specifically, we can update the root in $O(\log(|X|))$ operations using only the information about the node membership that one wants to replace and the current root. This update will allow an efficient on-chain update of the Merkle tree.

- $\mathsf{root}'_{dep}/\perp \leftarrow \textsc{Replace}(i, x, \mathsf{root}_{dep}, \mathsf{path}_i, x')$: takes as input the index i, the old element x and its membership proof path_i, and the new element, x' that we want to put in the i-th position. The algorithm verifies the membership of both x in the old root_{dep} using path_i, abort otherwise. Once the verification returns 1, it recomputes the root root'_{dep} using x' and path_i.

This efficient update is needed for the construction using verifiable computation.

3 On-chain Mixer System

This section presents the on-chain mixer system's components and the algorithms for the setup phase, the client, and the smart contract.

3.1 System Components

The system consists of two components: the *client* and the *smart contract*. A client controls blockchain addresses to interact with the smart contract, which governs a *pool* of assets. A client can either deposit/withdraw coins into/from the pool. The smart contract manages both deposit and withdrawal actions. The contract keeps track of various data structures and parameters to verify the validity of transactions that are sent to the contract.

3.2 Contract Setup

In the setup phase, all public parameters and the mixer smart contract are generated. The contract will be initialized with different data structures to avoid double withdrawal.

Furthermore, the deposit amount is specified as a fixed deposit amount of coins, amt. The smart contract is set up with two empty lists: *(i)* $\mathsf{DepositList}$, which includes all commitments cm contained in depositing transactions; *(ii)* $\mathsf{NullifierList}$, which contains all unique identifiers (i.e., sn) appeared in withdrawal transactions. We refer to pp^h as the state of the contract at block height h. The state includes all data structures of the contract, which were initialized in the setup phase. This state is implicitly provided to all client and contract algorithms. Finally, the smart contract is deployed on-chain in this phase.

3.3 Client Algorithm

A client can interact with the smart contract using the following algorithms. Note that each transaction is signed by the client using the private key associated with the blockchain address which issues the transaction.

- $(\mathsf{wit}, \mathsf{tx}_{dep}) \leftarrow \textsc{CreateDepositTx}(\mathsf{sk}, \mathsf{amt})$ takes a private key sk and an amount amt as inputs, and outputs a witness wit and a deposit transaction tx_{dep}.
- $\mathsf{tx}_{wdr} \leftarrow \textsc{CreateWithdrawTx}(\mathsf{sk}', \mathsf{wit})$ takes as input a private key sk' (which can be different from sk) and a witness wit, and outputs a withdrawal transaction tx_{wdr}.

3.4 Smart Contract Algorithm

The smart contract handles mixer deposits and withdrawals, with the following algorithms:

- $0/1 \leftarrow \textsc{AcceptDeposit}(\mathsf{tx}_{dep})$ takes as an input the deposit transaction tx_{dep}, and outputs 1 if the transaction was successful and 0 otherwise.
- $0/1 \leftarrow \textsc{IssueWithdraw}(\mathsf{tx}_{wdr})$ takes the withdrawal transaction tx_{wdr} as the input, and outputs 1 and transfers amt coins to the sender $\mathsf{tx}_{wdr}.\mathsf{sender}$ if the transaction was successful. Otherwise, the algorithm outputs 0.

3.5 System Goals

A secure on-chain mixer aims to satisfy the following properties.

Privacy. An on-chain mixer can break the linkability between user addresses. Consider an adversary with access to the entire history of all deposit and withdrawal transactions made to the mixer contract. Given a client who deposits into and withdraws from the mixer, the system ensures that the adversary cannot *(i)* link the deposit and withdrawal transactions issued by the client, and *(ii)* link the deposit and withdrawal addresses used by the client.

Correctness. A client cannot withdraw more coins from the contract than the client deposits. Moreover, a client cannot withdraw coins from the mixer prior to the deposit. The property prevents a client from stealing coins from the contract or other clients.

Availability. Clients should always be able to use the mixer. No entity can prevent clients from depositing or withdrawing coins.

Efficiency. The system should efficiently update clients' deposits and withdrawals, without causing expensive costs.

Fairness. Given a time interval, the costs of clients who deposit into (resp. withdraw from) the mixer during the interval remain approximately equal. This property allows for the amortization of costs over time, contributing to a balanced and equitable user experience.

4 Basic On-chain Mixer

4.1 Cryptographic Building Blocks

In the following, we present the cryptographic building blocks to construct on-chain mixers.

Deposit Commitments. Let $(\textsc{Com}, \textsc{Verify})$ be a commitment scheme that satisfies the hiding and binding properties. To deposit into the contract, a client samples two randomnesses k_{dep}, r, and computes the commitment $\mathsf{cm} = \textsc{Com}(m, r)$ as a part of a deposit transaction. In practice, the commitment scheme can be realized using a secure hash function $H_p : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{F}$.

Merkle Tree for Deposits. In an on-chain mixer, the leaves of the Merkle tree are initialized with zero values. The mixer smart contract preserves the Merkle tree T_{dep} of all deposit commitments. When deposit transactions occur, the smart contract keeps track of the total number of deposit transactions and updates the tree using the $\textsc{AcceptDeposit}$ algorithm. We define the Merkle proof for commitment cm_i as path_i. In addition, we define the root of the Merkle tree at block h as $\mathsf{root}_{wdr}^{\mathsf{curr}}$. We also denote root_{dep}.blockheight as the height of the blockchain block at the moment when root_{dep} is updated. Definition 4 formally defines a deposit Merkle tree. Note that $H_{2p} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ is a collision-resistant hash function.

▶ **Definition 4** (Deposit Merkle Tree). *A deposit Merkle tree encompasses three essential algorithms, namely $T.\textsc{Init}()$, $T.\textsc{Prove}()$, and $T.\textsc{Verify}()$, as defined in Definition 3. Additionally, the tree can be efficiently updated in a batch using the following algorithm:*

- $\mathsf{root}_{new} \leftarrow T.\textsc{Update}(Q)$ *takes a list Q containing new hashes as input. The algorithm inserts the elements in Q into the existing Merkle tree, and thus the new root will be updated. The output is the new root,* root_{new}.

Withdrawal Proof. To withdraw coins from the smart contract, a client needs to satisfy:

1. The client has committed values for certain existing deposit commitments utilized to construct the tree root using zk-SNARK.
2. The nullifier in the withdrawal transaction has not been used to withdraw previously.
3. The private key used to issue the withdrawal transaction is known by the client.

In short, a client needs to provide a proof showing the following statement for a Merkle tree T_{dep} with the root root_{dep}:

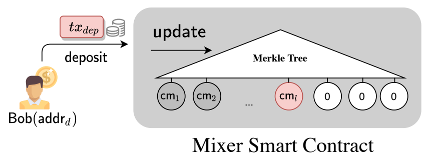
$$\begin{aligned} st_{wdr} : \{(\mathsf{pk}, \mathsf{sn}, \mathsf{root}_{dep}; \mathsf{sk}, k_{dep}, r, \mathsf{path}_i) : \mathsf{pk} = \textsc{extractPK}(\mathsf{sk}) \wedge \mathsf{sn} = H_p(k_{dep}, 0^\lambda) \wedge \\ \mathsf{cm} = \textsc{Com}(k_{dep}, r) \wedge T.\textsc{Verify}(i, \mathsf{cm}, \mathsf{root}_{dep}, \mathsf{path}_i))\} \end{aligned} \quad (1)$$

where $\mathsf{pk}, \mathsf{sn}, \mathsf{root}_{dep}$ are public values and $\mathsf{sk}, k_{dep}, r, \mathsf{path}_i$ are private values[2].

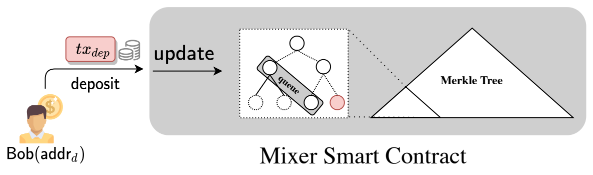
4.2 Workflow of Basic On-chain Mixer

Fig. 1a shows a basic solution for an on-chain mixer system, which is adopted by existing on-chain mixers, e.g., TC [4], Typhoon Network (TN) [6], and Cyclone [1]. In a nutshell, the high-level workflow of a basic on-chain mixer is as follows:

[2] sk is not needed in the private values if the zk-SNARK has simulation extractability property [8].



(a) Basic on-chain mixer. The Merkle tree is updated for every successful deposit.



(b) Improved on-chain mixer with MPB. The Merkle tree is updated once the leaves in the subtree corresponding to the deposit queue are full.

Figure 1 Overview of basic and improved on-chain mixer systems.

1. For each deposit transaction, a client adopts an address to issue a transaction, which transfers a *fixed* amount of coins into the mixer smart contract and generates a deposit commitment. The contract uses the Merkle trees of one pool to record the deposit commitments. Whenever a new deposit is made, the corresponding tree is updated, generating a new root. The recently updated roots are stored in a root list.
2. To withdraw, the client provides a withdrawal proof. The proof essentially shows that the client knows the secret to open a commitment in a deposit transaction and a membership proof by providing a path from the commitment to one root stored in the roots list (cf. Equation 1). The client will receive coins after the contract verifies the proof. The contract also records all withdrawal transactions' unique nullifiers.

4.3 System Goals of Basic On-chain Mixer

The basic mixers satisfy correctness, privacy, availability, and fairness when used properly:

Correctness. To issue a withdrawal, a client is required to present a proof associated with a unique nullifier. The mixer smart contract maintains a record of all unique nullifiers associated with withdrawals. Consequently, a client is restricted from issuing more withdrawals than the number of deposits they have previously conducted with the mixer.

Privacy. Considering the presence of n deposits within a mixer, a client's new deposit transaction becomes concealed amidst the n transactions sharing the same deposit amount. Furthermore, by employing separate deposit and withdrawal addresses, the probability of an adversary successfully linking the accurate deposit and withdrawal transactions is $\frac{1}{n}$.

Availability. As the mixer operates on a permissionless blockchain, which utilizes a global peer-to-peer network, adversaries are unable to impede clients from engaging with the mixer.

Fairness. The Merkle tree is updated for each deposit, resulting in equal gas fees for clients. When clients withdraw from the mixer, they incur costs for proof verification. These deposit and withdrawal costs are dependent on the gas price. Therefore, assuming the gas price does not fluctuate over a short timeframe, the mixer ensures fairness of costs.

However, we contend that the existing basic on-chain mixer design lacks efficiency since clients are required to pay a high deposit cost for updating the entire Merkle tree. In typical basic mixers such as TC, whenever a deposit request is made, the new coin is sequentially inserted into the Merkle tree, necessitating an update to the entire tree. This update operation becomes even more costly due to the utilization of Snark-friendly hash functions. For instance, by analyzing the 156,466 deposit transactions in the TC 0.1, 1, 10, and 100

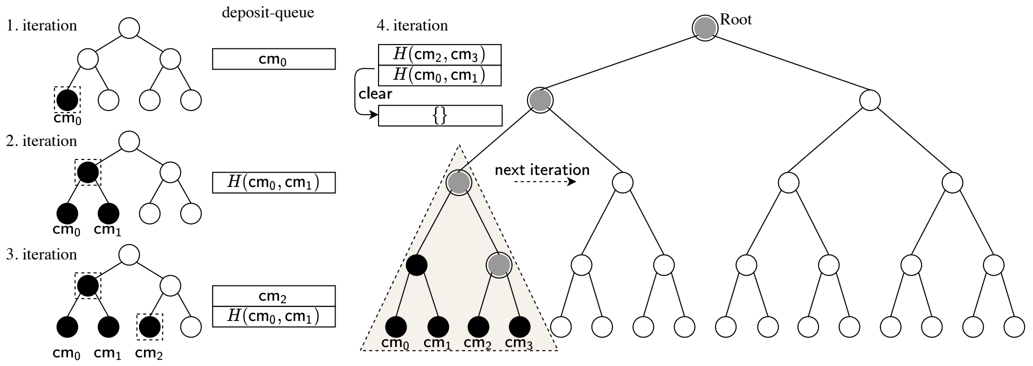

1. iteration
deposit-queue
cm_0
cm_0
2. iteration
$H(\mathsf{cm}_0, \mathsf{cm}_1)$
cm_0 cm_1
3. iteration
cm_2
$H(\mathsf{cm}_0, \mathsf{cm}_1)$
cm_0 cm_1 cm_2
4. iteration
$H(\mathsf{cm}_2, \mathsf{cm}_3)$
$H(\mathsf{cm}_0, \mathsf{cm}_1)$
clear
{}
Root
next iteration
cm_0 cm_1 cm_2 cm_3

Figure 2 Graphical illustration of the Merkle Pyramid Builder approach with a deposit queue size of four. cm_i represents the deposit commitment. In the first iteration, a deposit is made and included in the deposit queue. For subsequent deposits, the client combines the new deposit with the previously stored deposit, generating a new value that replaces the previous deposit in the queue. The second and third deposits are appended to the queue. Upon the fourth deposit, all hashes, including the root, are computed using all the values in the deposit queue. Finally, the deposit queue is cleared, and the process restarts from the beginning for the next deposit.

`ETH` pools from block 9,117,019 (December 16th, 2019) to 16,329,600 (January 3rd, 2023), we discovered that the average deposit cost for a TC `ETH` pool is approximately 1,111,030 gas, which is roughly 53 times higher than the Ethereum base fee of 21,000 gas.

5 Improving On-chain Mixers via Merkle Pyramid Builder

Existing on-chain mixers suffer from expensive deposit costs due to the frequent update of the Merkle tree per deposit. However, the one-deposit-one-update approach appears redundant since it is advisable to wait for other clients to deposit before initiating a withdrawal [38, 23]. To reduce the update time of the Merkle tree, we draw inspiration from the concept of Merkle tree mountain range [30, 33]. Accordingly, we propose a novel method called MPB.

Deposit-Queuing. To optimize the cost of deposits, we introduce a deposit-queuing method that batches transactions, resulting in less frequent updates of the Merkle tree.

▶ **Definition 5** (Deposit-Queuing). *A deposit-queuing method consists of three algorithms:*

- $q_{empty} \leftarrow \textsc{CreateQueue}(l)$ *takes as input an integer l, which specifies the number of deposit transactions to be batched. It returns an empty queue q_{empty} with a size of l, designed to store the nodes in a subtree with a height of $\log_2 l$. A deposit queue is considered full when all the leaves in the corresponding subtree are occupied.*
- $q' \leftarrow \textsc{Enqueue}(q, \mathsf{cm})$ *takes as input a queue q containing internal nodes of the subtree that can be used as helpers for an efficient update and a new commitment cm. As shown in Fig. 2, this procedure resembles the storage of internal nodes in the Merkle Mountain Range [33] and accumulator construction outlined in [30]. The function produces a new output queue q' containing internal nodes that enable the subtree to perform efficient updates during subsequent deposits.*
- $q_{empty} \leftarrow \textsc{ClearQueue}(q)$ *takes as input a queue q. The algorithm returns an empty queue corresponding to a new subtree.*

```
ContractSetUp($1^\lambda$)
1 : Sample $H_p : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{F}$ and $H_{2p} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$
2 : Choose $\mathsf{amt} \in \mathbb{Z}_{>0}$ to be a fixed deposit amount
3 : Choose $d \in \mathbb{Z}_{>0}$, Let $X = \{x_1, \ldots, x_{2^d}\}$ where $x_i = 0$ for all $x_i \in X$
4 : Initialize an empty tree $\mathsf{root}_{dep} = T.\textsc{Init}(1^\lambda, X)$,
5 : Choose $k \in \mathbb{Z}_{>0}$, set $\mathsf{RootList}_{wdr,k}[i] = \mathsf{root}_{dep}$, for $1 \le i \le k$
6 : Construct $C_{wdr}$ for statement described in Equation 1
7 : Let $\Pi$ be the zk-SNARK instance. Run $(\mathsf{ek}_{dep}, \mathsf{vk}_{dep}) \leftarrow \Pi.\textsc{Setup}(1^\lambda, C_{wdr})$
8 : Initialize: $\mathsf{DepositList} = \{\}, \mathsf{NullifierList} = \{\}, \mathsf{TotalFee} = 0$
9 : Initialize: $\mathsf{DepositQueue} \leftarrow \textsc{CreateQueue}(l)$
10 : Deploy smart contract with parameters :
     $\mathsf{pp} = (\mathbb{F}, H_p, H_{2p}, \mathsf{amt}, \mathsf{fee}_d, T, \mathsf{index}, \mathsf{RootList}_{wdr,k}, \mathsf{DespositQueue},$
          $(\mathsf{ek}_{dep}, \mathsf{vk}_{dep}), \mathsf{DepositList}, \mathsf{NullifierList}, \mathsf{TotalFee})$
```

Figure 3 Pseudocode for the smart contract setup in a mixer with MPB.

Merkle Pyramid Builder Approach. As shown in Fig. 1b, the key concept is to avoid frequent updates of the Merkle tree by aggregating deposit transactions and performing collective updates. In this approach, a deposit queue of size l corresponds to a subtree with a height of $\log_2 l$, as illustrated in Fig. 2. Notably, every even deposit in the sequence incurs no additional computational cost. However, each odd deposit requires hashing all the hashes up to the tree until no values remain on the left side within the same subtree. Finally, every l-th deposit necessitates computing all the hashes up to the root.

5.1 Contract Setup

The contract setup phase is to generate all the cryptographic parameters used in the protocol. In the setup phase (cf. Fig. 3), the algorithm ContractSetUp samples two secure hash functions H_p and H_{2p} from the collision-resistant hash families. The procedure further initializes amt as the fixed amount of coins that can be deposited into the mixer contract.

Setup Merkle Tree. We denote T as the deposit commitment Merkle tree with depth d. The algorithm T.Init initializes T as described in Section 4.1. The algorithm also initiates $\mathsf{RootList}_{wdr,k}$ to be the list of k most recent roots of T, which can address the concurrency issue for withdrawal transactions.

Setup zk-SNARK Parameters. We initialize a zk-SNARK instance Π with the algorithm Π.Setup with C_{wdr} as input, which can output two keys $(\mathsf{ek}_{dep}, \mathsf{vk}_{dep})$.

Setup Commitments and Nullifier Lists. The contract initializes two empty lists: *(i)* DepositList, which contains all cm included in deposit transactions; *(ii)* NullifierList, which contains all unique nullifiers sn committed in withdrawal transactions.

Setup Deposit Queue. The DepositQueue is initialized to track the number of deposit transactions in the queue and the number of transactions that have been hashed. This information determines whether the subsequent client needs to bear the cost of updating the Merkle tree. Note that the queue size $\log_2 l$ is the height of the subtree.

Client(sk, amt) $\xrightarrow{\mathsf{tx}_{dep}}$ **Smart Contract**

```
CreateDepositTx(sk, amt) :
1 :  Sample $(k_{dep}, r) \xleftarrow{\$} \{0,1\}^\lambda$
2 :  Compute $\mathsf{cm} = H_p(k_{dep}, r)$
3 :  return $\mathsf{wit} = (k_{dep}, r)$,
       $\mathsf{tx}_{dep} = (\mathsf{amt}, \mathsf{cm}, \mathsf{fee}_d)$
```

```
AcceptDeposit($\mathsf{tx}_{dep}$)
1 :  Parse $\mathsf{tx}_{dep} = (\mathsf{amt}', \mathsf{cm}, \mathsf{fee}'_d)$
2 :  Require: $\mathsf{amt}=\mathsf{amt}', \mathsf{fee}_d = \mathsf{fee}'_d, \mathsf{index} < 2^d$
3 :  Append cm to DepositList
4 :  Increment $\mathsf{index} = \mathsf{index} + 1$
5 :  DepositQueue = Enqueue(DepositQueue, cm)
6 :  $\mathsf{TotalFee} = \mathsf{TotalFee} + \mathsf{fee}_d$
     // The subtree is full
7 :  if DepositQueue is full:
8 :    Compute $\mathsf{root}_{new}$ from DespositQueue
9 :    ClearQueue(DespositQueue)
10 :   Append $\mathsf{root}_{new}$ to $\mathsf{RootList}_{wdr,k}$
      // Compensate the tree updater
11 :   Do $\mathtt{tx}_{dep}$.sender.transfer(TotalFee)
12 :   $\mathsf{TotalFee} = 0$
13 : else :
14 :   Enqueue(DespositQueue, cm)
15 : return 1
```

Figure 4 Deposit interactions between the client and the smart contract in a mixer with MPB. The computation of root_{new} (in line 8) can be done via the efficient replace function Def.2.2. We also note that line 3 is only for readability; in practice, users do not have to pay storage cost and can retrieve the list through emitted onchain events.

Functionality of $\mathsf{RootList}_{wdr,k}$. Similar to existing on-chain mixers [4, 23], our mixer contract maintains a list of the k most recent roots, denoted as $\mathsf{RootList}_{wdr,k}$. This list serves as an "AllowList" [16] to maintain a list of authorized parties (i.e., roots), which can address the concurrency issue.

Consider a scenario where user Bob attempts to withdraw using the most recently updated root $root_{k-1}$ from the $\mathsf{RootList}_{wdr,k}$. If an attacker, Alice, manages to make k deposits prior to Bob's withdrawal, these deposits would prompt k updates to the Merkle tree, effectively removing $root_{k-1}$ from the refreshed $\mathsf{RootList}_{wdr,k}$ and invalidating it for withdrawal.

However, as demonstrated in [23], this attack's cost is at least $k \times (\mathsf{amt} + \mathsf{fee})$, where amt represents the number of coins supported by the mixer pool (e.g., 0.1, 1, 10, and 100 for the TC `ETH` pools), and fee denotes the deposit fee. Moreover, the cost is higher in our improved mixer with MPB, as it typically requires $k \times l$ deposits to trigger k Merkle tree updates.

Importantly, the presence of the $\mathsf{RootList}_{wdr,k}$ does not introduce any vulnerability to double withdrawals. The only downside is that users might experience a marginally smaller anonymity set than the actual one. For instance, a user might use the root $root_0$ (corresponding to $|\mathsf{CmpSet}^{\mathsf{h}}|$ deposits), but due to concurrency, the latest root available may be $root_{k-1}$ (corresponding to $|\mathsf{CmpSet}^{\mathsf{h}}| + k$ deposits). In reality, $k \ll |\mathsf{CmpSet}^{\mathsf{h}}|$ (e.g., in TC, k is set to 100, while there are more than 26,000 deposits in each `ETH` pool [38]), rendering the difference in the anonymity set negligible.

5.2 Deposit Interaction

Fig. 4 shows the deposit process of a client in the mixer. This step allows the client to deposit coins into the mixer pool and obtain the witness for future withdrawal.

Client(sk)

```
CreateWithdrawTx(sk, wit) :
1 :  Parse $\mathsf{wit} = (k_{dep}, r)$
2 :  Obtain $\mathsf{pp}^h$ from the contract
3 :  Compute $\mathsf{sn}_{wdr} = H_p(k_{dep}, 0^\lambda)$
4 :  Compute $\mathsf{cm} = H_p(k_{dep}, r)$
5 :  Get index $i$ of cm from $\mathsf{DepositList}^h$
6 :  Choose $\mathsf{root}_{dep} \in \mathsf{RootList}_{wdr,k}$
7 :  Compute $\mathsf{path}^h_{dep,i}$ s.t.:
     $T.\textsc{Verify}(i, \mathsf{cm}, \mathsf{root}_{dep}, \mathsf{path}^h_{dep,i}) = 1$
8 :  Form $\mathsf{wit}_{dep} = (\mathsf{sk}, k_{dep}, r, \mathsf{path}^h_{dep,i})$
9 :  $\pi_{wdr} \leftarrow \Pi.\textsc{Prove}(\mathsf{ek}_{dep}, \mathsf{wit}_{dep},$
         $st[\mathsf{pk}, \mathsf{sn}_{wdr}, \mathsf{root}_{dep}])$
10 : return $\mathsf{tx}_{wdr} = (\mathsf{sn}_{wdr}, \mathsf{root}_{dep}, \pi_{wdr})$
```

$\xrightarrow{\mathsf{tx}_{wdr}}$

Smart Contract

```
IssueWithdraw(tx_wdr) :
1 :  Parse: $\mathsf{tx}_{wdr} = (\mathsf{sn}_{wdr}, \mathsf{root}_{dep}, \pi_{wdr})$
2 :  Require:
3 :    $\mathsf{root}_{dep} \in \mathsf{RootList}_{wdr,k}$
4 :    $\mathsf{sn}_{wdr} \notin \mathsf{NullifierList}$
5 :    $\Pi.\textsc{Verify}(\mathsf{vk}_{dep}, \pi_{wdr}, st[\mathtt{tx}_{wdr}.\mathsf{sender},$
       $\mathsf{sn}_{wdr}, \mathsf{root}_{dep}]) = 1$
6 :  Append $\mathsf{sn}_{wdr}$ to NullifierList
     // Send the original deposit back
7 :  Do $\mathtt{tx}_{wdr}$.sender.transfer(amt)
8 :  return 1
```

Figure 5 Withdrawal interactions between the client and the smart contract in a mixer with MPB. The state of the contract at block height h is denoted by pp^h. The withdrawal transaction tx_{wdr} contains the proof π_{wdr} that proves the client's knowledge of $\mathsf{cm} = H_p(k_{dep}, r)$ which is a valid member of the Merkle tree with the root root_{wdr}.

Client. A client deposits coins into the contract using the algorithm CreateDepositTx. The client first randomly selects two parameters k_{dep} and r, which are used to construct the commitment cm. The lengths of k_{dep} and r are defined and fixed during the initial setup. The deposit transaction tx_{dep} consists of the commitment cm and the coins amt that the client wants to deposit. Moreover, the client needs to specify the deposit fee fee_d in the transaction. To issue the transaction tx_{dep}, the client must use their private key sk to sign it. In addition, a witness wit is issued, which will be used to withdraw the coins in the future.

Contract. Upon receiving a deposit transaction tx_{dep}, the smart contract verifies that *(i)* the amount of coins and fees are as requested, and *(ii)* there is available space in the Merkle tree. If both conditions are met, the commitment in tx_{dep} is added to the DepositList. If the deposit queue is not full, the deposit fee will be added to the current total fee TotalFee. Otherwise, the total fee will be transferred to the last client and be reset as 0. With the algorithm Enqueue, the list DespositQueue will be updated. Moreover, if this deposit queue is full, the Merkle tree will be updated using the algorithm $T.\textsc{Update}$, and the smart contract will clear the deposit queue and finally add the new root to the list $\mathsf{RootList}_{wdr,k}$.

Note that the last client in a deposit queue will pay the gas cost for updating the Merkle tree. To guarantee fairness, the previous clients in the queue need to transfer additional deposit fees fee_d to the smart contract, which will be accumulated and serve as the reimbursement for the last client. A more comprehensive analysis of the deposit fee design will be presented in Section 6.4.

5.3 Withdrawal Interaction

Fig. 5 shows the mixer withdrawal interaction. This step allows the client to use the secret witness to generate a proof which is used to withdraw the corresponding deposited coins.

Client$(\mathsf{sk}, \mathsf{amt})$ $\xrightarrow{\mathsf{tx}_{dep}}$ **Smart Contract**

```
CreateDepositTx(sk, amt) :
1 : Sample $(k_{dep}, r) \xleftarrow{\$} \{0,1\}^{\lambda}$
2 : Compute $\mathsf{cm} = H_p(k_{dep}, r)$
3 : Read DepositQueue, $\mathsf{RootList}_{wdr,k}$
4 : $\mathsf{root}^{old}_{dep} = \mathsf{RootList}_{wdr,k}[-1]$
5 : Compute $\mathsf{path}_{\mathsf{index}}$ from DepositQueue
6 : $\mathsf{input} := (\mathsf{index}, 0, \mathsf{root}^{old}_{dep}, \mathsf{path}_{\mathsf{index}}, \mathsf{cm})$
7 : $\mathsf{root}_{new}, \pi_{dep} \leftarrow \textsc{VcProve}(\mathsf{ek}^{vc}_{dep}, \mathsf{input})$
8 : return $\mathsf{wit} = (k_{dep}, r)$,
    $\mathsf{tx}_{dep} = (\mathsf{amt}, \mathsf{root}_{new}, \pi_{dep}, \mathsf{cm}, \mathsf{fee}_d)$
```

```
AcceptDeposit(tx_dep)
1 : $\mathsf{tx}_{dep} = (\mathsf{amt}', \mathsf{root}_{new}, \pi_{dep}, \mathsf{cm}, \mathsf{fee}'_d)$
2 : Require:
      $\mathsf{amt} = \mathsf{amt}', \mathsf{fee}_d = \mathsf{fee}'_d, \mathsf{index} < 2^d$
3 : Append cm to DepositList
4 : $\mathsf{index} = \mathsf{index} + 1$
5 : $\mathsf{TotalFee} = \mathsf{TotalFee} + \mathsf{fee}_d$
6 : // The subtree is full
7 : if DepositQueue is full:
8 :   $\mathsf{root}^{old}_{dep} = \mathsf{RootList}_{wdr,k}[-1]$
9 :   Compute $\mathsf{path}_{\mathsf{index}}$ from DepositQueue
10 :  $\mathsf{input} := (\mathsf{index}, 0, \mathsf{root}^{old}_{dep}, \mathsf{path}_{\mathsf{index}}, \mathsf{cm})$
11 :  Require $\textsc{VcVerify}(\mathsf{vk}^{vc}_{dep}, \mathsf{input},$
       $\mathsf{root}_{new}, \pi_{dep}) = 1$
12 :  Append $\mathsf{root}_{new}$ to $\mathsf{RootList}_{wdr,k}$
      // Compensate the tree updater
13 :  Do $\mathsf{tx}_{dep}$.sender.transfer(TotalFee)
14 :  $\mathsf{TotalFee} = 0$
15 :  ClearQueue(DespositQueue)
16 : else :
17 :  Enqueue(DespositQueue, cm)
18 : return 1
```

Figure 6 Deposit interactions between the client and the smart contract in a mixer with VC.

Client. A client needs to create a withdrawal proof π_{wdr} with the secret witness wit, and the private key sk', to withdraw amt to the public key pk'. The proof π_{wdr} should be able to prove the three conditions mentioned in Section 4.1. The client then issues the withdrawal transaction tx_{wdr}, which consists of the nullifier sn_{wdr}, the root root_{dep}, and the proof π_{wdr}.

Contract. When receiving a withdrawal transaction $\mathsf{tx}_{wdr} = (\mathsf{sn}_{wdr}, \mathsf{root}_{dep}, \pi_{wdr})$, the contract checks the proof, π_{wdr}, and confirms that the nullifier sn_{wdr} is not in the NullifierList. The contract then appends sn_{wdr} to NullifierList to avoid future double withdrawals. Finally, the smart contract transfers amt coins to the withdrawal address specified by the client.

5.4 Further Improvement with Verifiable Computation Techniques

In this section, we employ verifiable computation techniques [17, 22] to enhance the efficiency of deposit costs. In an on-chain mixer utilizing VC, the key idea is to conduct the computation off-chain. In this approach, the client evaluates the Merkle tree root and transmits both the result and a proof of correct computation to the smart contract. The contract subsequently verifies the validity of the proof to ensure the accuracy of the computation.

5.4.1 Building Blocks for On-Chain Mixer with VC

Verifiable Computation Scheme. In a VC scheme (cf. Definition 6), the verifier selects a function and an input to send to the prover. The prover evaluates the function on the input and returns the result, along with proof attesting to the result's validity. The verifier then verifies that the output provided by the prover indeed corresponds to the result of the function evaluated on the given input. The objective is to achieve efficient verification, significantly faster than the actual computation of the function itself.

▶ **Definition 6** (Verifiable Computation Scheme). *Let f be a function, expressed as an arithmetic circuit over a finite field $\mathbb{F}$, and let λ be a security parameter.*

- $(\mathsf{ek}, \mathsf{vk}) \leftarrow \textit{VcInit}(1^\lambda, f)$ *takes a security parameter and an arithmetic circuit as input and generates two public keys: an evaluation key* ek *and a verification key* vk*.*
- $(y, \pi) \leftarrow \textit{VcProve}(\mathsf{ek}, x)$ *takes as input an element x and the evaluation key* ek *and computes $y = f(x)$ and a proof π that y has been correctly computed.*
- $0/1 \leftarrow \textit{VcVerify}(\mathsf{vk}, x, y, \pi)$ *takes the verification key* vk*, the input/output (x, y) of the computation f and the proof π and outputs 1 if $y = f(x)$ and 0 otherwise.*

Any SNARK instance (e.g., Groth16 [19]) can be used to construct a VC scheme.

Deposit Proof. To deposit coins to the smart contract, a client needs to give a proof showing the following relation for a Merkle tree T with a root root_{dep}:

$$st_{dep} : \{\mathsf{pk}, \mathsf{cm}, \mathsf{root}_{dep}^{old}, \mathsf{root}_{dep}^{new}, \mathsf{path}_i : \mathsf{root}_{dep}^{new} = T.\textsc{Replace}(\mathsf{index}, 0, \mathsf{root}_{dep}^{old}, \mathsf{path}_i, \mathsf{cm})\} \quad (2)$$

In this construction, the function f that we want to verify is the replacement algorithm (i.e., Replace) defined for the Merkle tree.

Concurrent Updating Operations. During the construction process using VC, it is possible to encounter a situation where two users are simultaneously updating the tree. However, this does not present a problem. Both deposits will be processed since the smart contract only accepts the VC proof when the queue is full. Consequently, only one user (the faster one) is required to pay and is subsequently compensated for the computation cost. Meanwhile, the other deposit will be placed as the first element of the queue.

5.4.2 Algorithms for On-Chain Mixer with VC

Contract Setup. The ContractSetUp algorithm of a mixer with VC differs from the MPB method (cf. Section 5.1): An additional instance must be initialized for the verifiable computation (cf. Section 5 in the full version of this paper [39]). The rest remains unchanged. In particular, we initialize a verifiable computation instance Π_{dep} with the algorithm $\Pi_{dep}.\textsc{VcInit}$ with C_{dep} as input. We obtain two keys $(\mathsf{ek}_{dep}^{vc}, \mathsf{vk}_{dep}^{vc})$.

Deposit Interaction. Fig. 4 shows the deposit interaction between a client and the mixer with VC, in which the client computes the updated root when depositing into the mixer.

Client. The only difference between a mixer with the MPB and the one with VC is that the last client must compute and prove the valid computation of the new root in addition to cm and amt. The rest remains unchanged.

Contract. In contrast to MPB, the smart contract now performs fewer calculations. The smart contract checks the client's proof of the validity of the new root. If the proof is valid, the root of the contract is modified to match the client's root.

Withdraw Interaction. Because the verifiable computation only alters the deposit method, the way to withdraw coins in a mixer stays the same as in Section 5.1.

6 System Analysis

In this section, we prove that our improved mixers with MPB and VC both can guarantee *privacy*, *correctness*, *availability*, and *fairness*, while they achieve different degrees of *efficiency*.

6.1 Privacy

6.1.1 Linking deposit and withdrawal transactions

Using a similar definition proposed in the AMR system [23], we first investigate the potential for an adversary to establish a link between a withdrawal transaction and its associated deposit transaction. We denote h as the height of the blockchain. Given a block height h, we define $\mathsf{CmpSet}^{\mathsf{h}}$ as the set of commitments of deposits made by honest users within a mixer pool. Within this context, for a given $\mathsf{cm} \in \mathsf{CmpSet}^{\mathsf{h}}$, we denote $\mathsf{tx}_{dep}(\mathsf{cm})$ as the deposit transaction which includes cm. Given a deposit transaction tx_{dep} including the commitment cm, and a withdrawal transaction tx_{wdr} with the nullifier sn, we say sn is originated from cm (denoted as $\mathsf{sn} \overset{origin}{\leftarrow} \mathsf{cm}$) if $\exists(k, r) \in \{0,1\}^{\lambda}$, s.t., $\mathsf{cm} = \textsc{Com}(k, r) \land \mathsf{sn}_{wdr} = H_p(k, 0^{\lambda})$. Therefore, the adversarial advantage of linking the withdrawal transaction tx_{wdr} to its corresponding deposit transaction tx_{dep}, can be quantified as the probability that an adversary correctly guesses the commitment that originates the nullifier value in tx_{wdr} (cf. Definition 7).

▶ **Definition 7** (Adversarial Advantage in Transaction Linking). *Let $\mathcal{A}$ be a PPT adversary, and $\mathsf{tx}^{\mathsf{h}}_{wdr}$ be a valid withdrawal transaction issued at block h by an honest user. Let $\mathsf{sn}^{\mathsf{h}}_{wdr}$ be the nullifier included in $\mathsf{tx}^{\mathsf{h}}_{wdr}$. We define the adversarial advantage as follows:*

$$\mathsf{Adv}^{\mathsf{h}}_{\mathcal{A},tx} = \Pr[\mathcal{A}(\mathsf{tx}^{\mathsf{h}}_{wdr}) \rightarrow \mathsf{tx}_{dep}(\mathsf{cm}),\ \ s.t.\ \ \mathsf{cm} \in \mathsf{CmpSet}^{\mathsf{h}} \land \mathsf{sn}^{\mathsf{h}}_{wdr} \overset{origin}{\leftarrow} \mathsf{cm}]$$

6.1.2 Linking deposit and withdrawal addresses

In practice, a user may utilize the same address to make multiple deposits or withdrawals within a mixer pool [38]. Rather than linking individual deposit and withdrawal transactions, the adversary may choose to target the linkability between addresses, specifically linking deposit and withdrawal addresses controlled by the same user.

Given a block height h, we denote $\mathsf{DepAddrSet}^{\mathsf{h}}$ as the set of addresses that are used to deposit coins into a mixer pool from honest users, and $\mathsf{WdrAddrSet}^{\mathsf{h}}$ as the set of withdrawal addresses of honest users.

Given a deposit address $\mathsf{addr}_d \in \mathsf{DepAddrSet}^{\mathsf{h}}$ and a withdrawal address $\mathsf{addr}_w \in \mathsf{WdrAddrSet}^{\mathsf{h}}$, we say addr_d and addr_w are linked if they belong to the same user, i.e., the user controls both the private keys of addr_d and addr_w. We denote this as $\mathsf{addr}_d \overset{link}{\leftrightarrow} \mathsf{addr}_w$.

Therefore, the adversarial advantage of linking the withdrawal address addr_w to the corresponding deposit address addr_d, is the probability that an adversary can correctly determine if they are controlled by the same user (cf. Definition 8).

▶ **Definition 8** (Adversarial Advantage in Addresses Linking). *Let $\mathcal{A}$ be a PPT adversary, and $\mathsf{addr}_w \in \mathsf{WdrAddrSet}^{\mathsf{h}}$ be an address that has been previously used to withdraw coins from the mixer pool before the block height h. We define the adversarial advantage as follows:*

$$\mathsf{Adv}^{\mathsf{h}}_{\mathcal{A},addr} = \Pr[\mathcal{A}(\mathsf{addr}_w) \rightarrow \mathsf{addr}_d,\ \ s.t.\ \ \mathsf{addr}_d \in \mathsf{DepAddrSet}^{\mathsf{h}} \land \mathsf{addr}_d \overset{link}{\leftrightarrow} \mathsf{addr}_w]$$

6.1.3 Privacy Analysis

We present the following claims to analyze the adversarial advantages of linking transactions and addresses in our improved mixers. For detailed proofs, we refer the reader to the full version of this paper [39].

▷ Claim 9. Under the assumption that all underlying cryptographic primitives are secure, the adversarial advantage in linking a withdrawal transaction at block height h to the corresponding deposit transaction (cf. Definition 7) satisfies: $\mathsf{Adv}^{\mathsf{h}}_{\mathcal{A},tx} \leq \frac{1}{|\mathsf{CmpSet}^{\mathsf{h}}|} + \mathsf{negl}(\lambda)$.

▷ Claim 10. Under the assumption that all underlying cryptographic primitives are secure, the adversarial advantage in linking a withdrawal address at block height h to the corresponding deposit address (cf. Definition 8) satisfies: $\mathsf{Adv}^{\mathsf{h}}_{\mathcal{A},addr} \leq \frac{1}{|\mathsf{DepAddrSet}^{\mathsf{h}}|} + \mathsf{negl}(\lambda)$.

▶ Remark. Note that the increase in deposit transactions will not always decrease the adversarial advantage in linking deposit and withdrawal addresses, because an address can be used to deposit multiple times in a mixer pool. We also remark that the deposits in the queue are not considered in our privacy analysis, because they are not finalized and do not contribute to the set of commitments $\mathsf{CmpSet}^{\mathsf{h}}$ and the set of deposit addresses $\mathsf{DepAddrSet}^{\mathsf{h}}$.

6.2 Correctness

In the following, we prove that our system can guarantee correctness by demonstrating that the probability that an adversary can withdraw more times than deposits is negligible.

Consider an adversary who deposits into the mixer via a transaction $\mathsf{tx}^{\mathsf{h}}_{dep}$ at block h, and the transaction includes the commitment cm. We define the adversarial advantage of double withdrawing as the probability that the adversary can generate two withdrawal transactions linking to $\mathsf{tx}^{\mathsf{h}}_{dep}$ (cf. Definition 11).

▶ **Definition 11** (Adversarial Advantage in Double Withdrawing). *Let $\mathcal{A}$ be a PPT adversary, which issues a deposit transaction tx^{h}_{dep} at block h. Let cm be the commitment included in tx^{h}_{dep}. We define the adversarial advantage as follows:*

$$\mathsf{Adv}^{\mathsf{h}}_{\mathcal{A},ww} = \Pr[\mathcal{A}(tx^{\mathsf{h}}_{dep}(\mathsf{cm})) \to \left(tx^{0}_{wdr}\left(\mathsf{sn}^{\mathsf{h}_0}_{wdr}\right), tx^{1}_{wdr}\left(\mathsf{sn}^{\mathsf{h}_1}_{wdr}\right)\right)$$
$$s.t.\ \mathsf{sn}^{\mathsf{h}_0}_{wdr} \overset{origin}{\leftarrow} \mathsf{cm} \wedge \mathsf{sn}^{\mathsf{h}_1}_{wdr} \overset{origin}{\leftarrow} \mathsf{cm} \wedge \mathsf{h}_0 > \mathsf{h} \wedge \mathsf{h}_1 > \mathsf{h} \wedge \mathsf{sn}^{\mathsf{h}_0}_{wdr} \neq \mathsf{sn}^{\mathsf{h}_1}_{wdr}]$$

Intuitively, double withdrawals occur when the adversary manages to generate two different nullifiers, $\mathsf{sn}^{\mathsf{h}0}_{wdr}$ and $\mathsf{sn}^{\mathsf{h}_1}_{wdr}$, both originating from the same commitment cm. However, the probability of this event is negligible (cf. Claim 12).

▷ Claim 12. Assuming that all underlying cryptographic primitives are secure, the adversarial advantage in successfully generating two distinct withdrawal transactions which correspond to the same deposit transaction (cf. Definition 11) satisfies: $\mathsf{Adv}^{\mathsf{h}}_{\mathcal{A},ww} \leq \mathsf{negl}(\lambda)$.

6.3 Availability

Our system, comprising either MPB or VC, ensures availability. Similar to existing on-chain mixers such as TC [4] on Ethereum and TN [6] on BSC, our improved mixer can operate autonomously on smart-contract-enabled blockchains. It should be noted that a centralized regulator could affect the availability of an on-chain mixer. For instance, the regulator can impose sanctions [36] on on-chain mixers and require decentralized application frontends or Front-running as a Service (e.g., Flashbots MEV-boost relays) to censor mixer-related

Table 1 Deposit cost calibration over time. The first $l-1$ clients' cost in the l_i-th deposit queue is calibrated by the total cost in the previous queue.

deposit queue		l_{i-1}	l_i
Cost	first $l-1$ clients	d^0_{i-1}	$d^0_i = \frac{C_{i-1}}{l}$
	last client	d^1_{i-1}	d^1_i
	total	$C_{i-1} = (l-1) \cdot d^0_{i-1} + d^1_{i-1}$	$C_i = (l-1) \cdot d^0_i + d^1_i$

transactions [14]. However, users still have the option to utilize the command line interface or intermediary addresses to bypass the censorship and interact with the on-chain mixer smart contracts [40].

6.4 Fairness

To ensure fairness, our improved mixer employs a mechanism that ensures nearly equal fees are paid by all l clients in the same deposit queue when updating the Merkle tree. The approach involves each client in the queue, except the last one, paying an additional gas fee of d^0 in their deposit transactions to the smart contract. The contract then keeps track of the accumulated fees, totaling $(l-1) \cdot d^0$. When it is the turn of the l-th client to update the Merkle tree and clear the queue, the client will be responsible for paying the remaining gas fees, denoted as d^1. Note that gas prices can vary over time, which means that the cost for each client needs to be adjusted accordingly. To achieve this, the payment fees for clients in the l_i-th deposit queue are calibrated based on the total cost of the l_{i-1}-th queue, where $i \geq 1$. Specifically, if the total update cost of the l_{i-1}-th queue is C_{i-1}, then the average cost for the previous $l-1$ clients in the l_i-th deposit queue is calculated as $\frac{C_{i-1}}{l}$. Table 1 illustrates the deposit costs for clients in a queue. It is worth noting that while the cost of the last client technically differs slightly from that of the remaining $l-1$ clients, we can prove that the improved mixer achieves fairness through the following analysis.

We first provide the definition of fairness for an on-chain mixer.

▶ **Definition 13** (ϵ-Fairness). *Given two blocks b_0, b_1, and their corresponding gas prices* $\textsc{price}(b_0)$ *and* $\textsc{price}(b_1)$*, we say a mixer achieves ϵ-fairness if the deposit costs* $\textsc{cost}(b_0)$ *and* $\textsc{cost}(b_1)$ *in blocks b_0 and b_1 satisfy* $|\textsc{cost}(b_0) - \textsc{cost}(b_1)| \leq \epsilon \cdot |\textsc{price}(b_0) - \textsc{price}(b_1)|$.

We proceed to prove that our improved mixer can achieve ϵ-fairness when adopting the deposit cost calibration design in Table 1.

▷ Claim 14. Given a timeframe T covers at least two deposit queues l_{i-1} and l_i, whose final deposits occur in block $b_{l_{i-1}}$ and b_{l_i} respectively. Assuming that the deposit gas for updating the Merkle tree in different queues is constant, denoted as gas_{update}, then our improved mixer can achieve ϵ-fairness in the timeframe, where $\epsilon = gas_{update}$.

6.5 Efficiency

6.5.1 Efficiency for On-Chain Mixer with MPB

Compared to the basic on-chain mixer design (cf. Section 4), our improved mixer system offers enhanced efficiency. We consider a deposit Merkle tree with a size of n. In a basic on-chain mixer, the computation cost for l deposits is $l \cdot O(\log(n))$ as the entire Merkle tree for each deposit needs to be updated. However, in our improved mixer design with an l-length deposit queue, the Merkle tree only needs to be updated once per l deposits. Therefore, the computation cost for l deposits is $l + O(\log(n))$.

We further define the ratio as $\gamma = \frac{l+O(\log(n))}{l\cdot O(\log(n))}$ to quantify the times of the saving deposit cost in our improved mixer compared to the basic mixer solution. Note that the deposit queue size l is much smaller than the Merkle tree size n; therefore, $\gamma = \frac{l}{O(\log(n))} + \frac{O(\log(n))}{l} \approx \frac{O(\log(n))}{l}$. The larger the queue size l, the more deposit cost clients can save. However, the finalization time of the deposit is also increasing over l. We should properly choose the size l to deal with the trade-off between the deposit cost and finalization time. We will provide the detailed evaluation results in Section 7.2.

6.5.2 Efficiency for On-Chain Mixer with VC

Thanks to VC, the deposit cost in a mixer can further be reduced. Consider that the computation cost of verifying an updated Merkle tree root is costVc, and the cost of generating an updated Merkle tree root is costGc. Therefore, in a basic on-chain mixer, the computation cost for l deposits is $l \cdot O(\log(n)) \cdot \mathsf{costGc}$ as the Mekle tree is updated per deposit. The computation cost for l deposits in an improved mixer with VC is $l+O(\log(n))\cdot\mathsf{costVc}+\mathsf{costGc}$. The deposit cost saving is $\gamma^{vc} = \frac{l+O(\log(n))\cdot\mathsf{costVc}+\mathsf{costGc}}{l\cdot O(\log(n))\cdot\mathsf{costGc}} = \frac{l}{O(\log(n))\cdot\mathsf{costGc}} + \frac{\mathsf{costVc}+\frac{\mathsf{costGc}}{O(\log(n))}}{l\cdot\mathsf{costGc}} \approx \frac{\mathsf{costVc}+\frac{\mathsf{costGc}}{O(\log(n))}}{l\cdot\mathsf{costGc}}$. Note that in verifiable computations, the computation cost of verification costVc is inferior to the cost of proof generation costGc. Therefore, an on-chain mixer with verifiable computations can reduce deposit costs more than one with MPB. Section 7.4 will provide more quantification results.

7 Evaluation

In this section, we implement our improved mixer designs and evaluate their performance.

Cryptographic Primitives. We adopt Groth's zk-SNARK, Groth16 [19], as our instance of zk-SNARK owing to its efficiency in terms of its proof size and the calculations required by the verifier. Although the original Groth16 does not have simulation intractability proof, it is recently proven to achieve weak simulation extractability [8]. We elaborate more on the discussion in the full version of this paper [39]. We employ the Pedersen hash function [26] for H_p and the MiMC hash function [7] for H_{2p}, as cryptographic hash functions. Compared to arithmetic circuits that rely on other hash functions, such as Jubjub [3], arithmetic circuits that employ MiMC hash can produce a smaller number of constraints and operations. In addition to being created exclusively for SNARK applications, MiMC hash functions are also very gas-efficient for Ethereum smart contract applications.

Software Setup. For the arithmetic circuit design, we leverage the Circom library [9] to build the withdrawal circuit, C_{wdr}, for the relation specified in Equation 1. We utilize Groth16[19] proof system implemented by the snarkjs package [10] to construct the client's algorithms.. We establish a trusted environment for evaluating the mixer smart contract and clients. We deploy the on-chain mixer system on the EVM ganache [2].

Hardware Setup. We perform our experiment on a standard desktop system with an 11th Gen Intel(R) Core(TM) i7-11800H with 2.30G CPU and 16GB RAM in configuration.

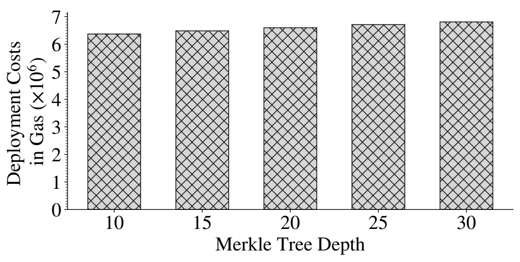


Figure 7 Cost of deploying the MPB smart contract with different Merkle tree sizes.

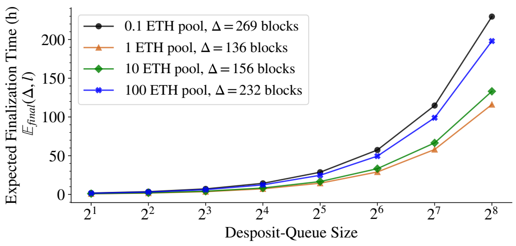


Figure 8 Expected deposit finalization time in TC `ETH` pools with various deposit queue sizes.

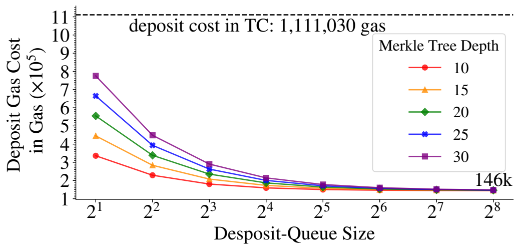


Figure 9 Average deposit costs per client for various queue sizes and Merkle tree depths. The deposit cost in TC (dashed line) is $\approx 1.1M$ gas.

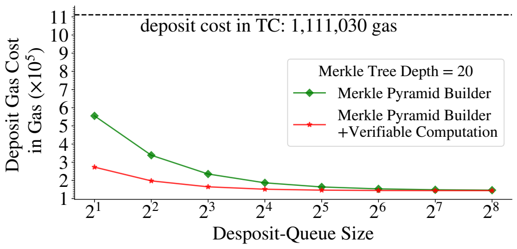


Figure 10 Average on-chain deposit costs per client for various deposit queue sizes. The VC approach can further reduce the deposit cost.

7.1 Evaluating Merkle Pyramid Builder Costs

We evaluate the performance of the MPB system using different configurations of Merkle tree depths (i.e., $d = 10, 15, 20, 25, 30$) and deposit queue lengths ($l = 2, 4, 8, 16, 32, 64, 128$).

On-chain Deployment Costs. Fig. 7 shows the expense associated with deploying smart contracts. The Merkle tree depth does not have a huge influence on the deployment costs: the total cost is always between $6M$ and $7M$ gas. However, the deployment cost is a one-time expense that can be amortized over the duration of the contract.

On-chain Deposit Costs. Fig. 9 provides a visualization of the cost reduction achieved by increasing the size of the deposit queue. We observe that after reducing or raising the depth of the Merkle tree, the cost decreases or increases, accordingly. Note that reducing the depth of the Merkle tree reduces the number of users, while raising the depth significantly increases the time required to compute the withdrawal proof. We can also observe that for a deposit queue of length 128, the gas costs are around $146K$, which is consistent amongst the various Merkle tree depths. Figure 9 also shows that this cost is merely $\frac{1}{7}$ of the cost of depositing in TC. Note that the costs associated with the deposit queue with a size of one, would roughly correspond to those of TC.

7.2 Evaluating Deposit Finalization Time

The deposit queue sizes will affect the deposit finalization time, i.e., the time that users need to wait for their deposits to be updated in the Merkle tree. To quantify the waiting time, we crawl the historical deposits in the TC four ETH pools (i.e., 0.1, 1, 10, and 100 ETH pools) over time. As shown in the full version of this paper [39], from October 1st, 2020 to

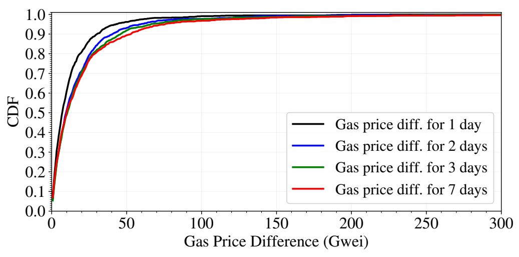


Figure 11 Distribution of gas price differences for TC deposit transactions.

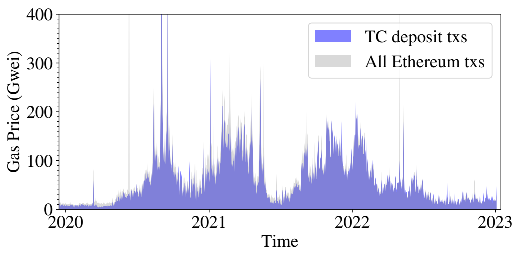


Figure 12 Average gas price of TC deposit transactions over time.

August 8th, 2022 (i.e., the date when Office of Foreign Assets Control (OFAC) announced the sanctions against TC), the average number of daily deposits in the four pools is 51 ± 25. Even after the OFAC sanctions' announcement, the average number is still 9 ± 8. Therefore, when the deposit queue size is set as 32, users merely need to wait for less than 4 days on average to ensure their deposits are finalized in a TC pool adopting our MPB method.

To further quantify the expected finalization time, we define Δ as the average timeframe between two deposits. Therefore, the expected finalization time $\mathbb{E}_{final}(\Delta, l)$ of a client in a deposit queue with the size l is $\mathbb{E}_{final}(\Delta, l) = \sum_{i=1}^{l} \frac{1}{l} \cdot (l - i) \cdot \Delta = \frac{(l-1)\cdot\Delta}{2}$.

Based on our empirical data, we can calculate that the average timeframes between two deposits in the TC 0.1, 1, 10, and 100 are 269, 136, 156, and 232 blocks respectively. Fig. 8 shows the expectation of the deposit finalization time when choosing different deposit queue sizes for TC ETH pools. Note that we set the block timeframe as $12s$, which corresponds to the slot time in the post-merge Ethereum. The results indicate that the expected deposit finalization time increases linearly over the deposit queue size.

7.3 Evaluating Deposit Gas Prices

In the following, we leverage the gas prices of TC historical deposit transactions to quantify our improved mixer's fairness.

Fig. 12 shows the average gas price of TC deposit transactions and all Ethereum transactions over time. We observe that generally, the TC deposit transactions have almost the same average gas price as other Ethereum transactions. To further measure the variance of gas price, we calculate the differences between sequential daily deposits in TC `ETH` pools. As shown in Fig. 11, we plot the distribution of the gas price differences for the deposits in sequential n days, where $n = 1, 2, 3$ and 7. We observe that more than 80% gas price differences in $n (n \leq 7)$ days are inferior to 30 GWei. Therefore, if we set the expected deposit finalization time as less than 7 days, the deposit cost differences between the last client and other clients in the same queue are inferior to 30 GWei $\times$ gas_{update}. Recall that gas_{update} is the total gas consumed for updating the Merkle tree in an on-chain mixer (cf. Section 6.4).

When setting the deposit queue size as 2^5, and the Merkle tree depth as 20, we have $gas_{update} = 2^5 \times 1.78 \times 10^5 = 5.69M$ gas (cf. Fig. 9). We can further calculate that the cost difference is $5.69 \times 10^6 \times 30 \times 10^9$ Wei $= 1.7 \times 10^{17}$ Wei $= 0.17$ `ETH`.

7.4 Evaluating Verifiable Computation Costs

TC supports anonymity mining [35] to incentivize users to user their `ETH` mixer pool. In order to claim the anonymity mining rewards, users can provide the Merkle tree roots to show their deposit and withdrawal transactions have already been successfully performed.

This design is similar to our deposit with off-chain proof generation. Therefore, to evaluate the deposit cost of VC, we adopt a similar circuit of TC anonymity mining[3] to implement the VC deposit contract and test it locally. Our evaluation shows that the average deposit cost of VC is approximately $436K$ gas. Thus, the saving deposit cost ratio of a client in a deposit queue of size l can be calculated as $\frac{1M}{436K \cdot l} = \frac{2.29}{l}$.

Note that in the improved mixer with VC, the smaller the queue size (i.e., l) is, the more cost per client can save (cf. Fig. 10). Therefore, when combining VC and MPB approaches, our improved mixer enables a significant improvement in the cost with a small deposit queue size (e.g., 4 or 8), which allows a short waiting time for deposit finalization.

8 Related Work

Blockchain Add-On Privacy Solutions. To enhance the privacy of non-privacy-preserving blockchains such as Bitcoin and Ethereum, numerous mixers are proposed in academic works or deployed in practice. For instance, Meiklejohn *et al.* [25] propose a smart-contract-based mixer named Möbius, which adopts linkable ring signatures and stealth addresses [28] to hide the addresses of transaction senders and recipients. However, the anonymity set size of Möbius is limited by the ring size, and the withdrawal cost increases linearly with the ring size. CoinJoin [24] is a Bitcoin mixer, which enables a user can collaborate with others to merge multiple transactions, thereby breaking the linkability between addresses. On-chain mixers [4, 6, 5, 1] are inspired by Zerocash [32] and are running on smart-contract-enabled blockchains to obfuscate the link between the users' deposit and withdrawal using ZKPs. Le *et al.* propose an on-chain mixer design [23], which incentivizes users to participate in a mixer. Shortly after [23], TC follows by adding anonymity mining as a deposit reward scheme for attracting users [35]. In addition to blockchain add-on privacy solutions, several existing works have proposed privacy-enhanced and regulated solutions for Central Bank Digital Currencies (CBDCs). These solutions include UTT [34], PEReDi [21], and Platypus [43]. Despite their theoretical promise, the practicality of implementing and running these solutions efficiently on the Ethereum platform remains unclear.

Blockchain Mixer Analysis. Although mixers can break the linkability among user addresses by design, in practice, mixers are not being used properly. Wu *et al.* [42] propose a generic abstraction model for Bitcoin mixers. They identify two mixing mechanisms, i.e., swapping and obfuscating, and present a method to reveal mixing transactions that leverage the obfuscating mechanism. Through analyzing the mixing activities in TC [4] and TN [6], Wang *et al.* [38] proposes five heuristics to link the deposit and withdrawal addresses of on-chain mixers. Their heuristics can reduce a mixer's anonymity set size by more than 34.18%. Moreover, Wang *et al.*'s measurement work [38] also indicates that the reward mechanism of on-chain mixers tends to attract profit-driven but privacy-ignorant users.

9 Discussion

Trade-Off Between Latency and Cost. In our MPB mixer design, typically, the last depositor in a deposit queue initiates the Merkle tree update when the queue is full. However, we recognize the potential for including an optional function that allows any user to pay and trigger this update. This functionality would enable users to achieve faster-verified deposits

[3] `https://github.com/tornadocash/tornado-anonymity-mining`

by updating the tree without waiting for the queue to be filled. In such cases, the tree updater must strike a balance between latency and cost since the accumulated fees in an unfilled queue may not fully compensate for the updater's expenses.

Trade-Off Between Storage and Computation. In MPB, the subtree is updated with every deposit to optimize on-chain storage costs while maintaining a balance between storage and update efficiency. Specifically, in MPB, the total computation cost of updating a queue with l deposits is $l + O(\log(n))$, and the storage cost is $O(\log l)$. An alternative approach is to use a more "naive" batch update, where the mixer contract keeps a record of all l deposits in the queue and updates the subtree per queue. In this case, the total computation cost is still $l + O(\log(n))$, and the storage cost is $O(l)$.

Comparison with Other Data Structures. In addition to MPB, we have also conducted tests on other data-authenticated structures, such as dynamic RSA accumulators [11, 27, 12]. Accumulators offer the capability to generate proofs that demonstrate the membership of potential items in a specific set. One notable advantage of RSA accumulators is their efficiency in adding new items to an existing set. However, we observe that they do not effectively reduce deposit appending costs, as the mixer contract is required to perform expensive primality testing to identify prime numbers [20] as commitments on-chain. Furthermore, the process of proving membership in zero knowledge will incur significantly higher expenses.

Weak Simulation Extractability of Groth16. It is well-known that Groth16 zk-SNARK is malleable. Hence, one needs to use a deterministic nullifier to prevent double withdrawing. Also, in our design (refer to Eq. 1), it is necessary for a user to supply the associated private key sk as part of the private input (i.e., witness). This measure is taken to mitigate the risk that an adversary can replace a public key with his public key. However, in practice, we only need to use the public as a parameter to the public input. This is considered secure, given that Groth16 is recently proven to have the weak simulation extractability property [8]. This means that an adversary cannot produce a valid proof for a different input instance. Adopting this approach can notably reduce the proof generation costs for provers.

10 Conclusion

This paper investigates how to reduce the deposit cost of on-chain mixers. We first propose a design named MPB, which batches deposits in a queue and updates the Merkle tree per batch. This methodology reduces the Merkle tree update times and, thus, decreases the deposit cost. Specifically, our evaluation results show that MPB can achieve 7× fewer costs for depositing than state-of-the-art on-chain mixers. Moreover, we leverage off-chain verification to reduce the cost further. We also prove that our improved on-chain mixer designs can guarantee *correctness*, *privacy*, *availability*, *efficiency*, and *fairness*. We hope our work can engender further research into more secure and user-friendly on-chain mixers.

References

1 Cyclone. Available at: `https://cyclone.xyz/bsc`.
2 Ganache. Available at: `https://trufflesuite.com/ganache/`.
3 Jubjub. Available at: `https://z.cash/technology/jubjub/`.
4 Tornado cash. Available at: `https://tornado.cash/`, before August 8th, 2022.
5 Typhoon.cash. Available at: `https://typhoon.cash/`.

6 Typhoon.network. Available at: `https://app.typhoon.network/`.

7 Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 191–219. Springer, 2016.

8 Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-snark. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I 25*, pages 457–475. Springer, 2021.

9 Jordi Baylina, Kobi Gurkan, Roman Semenov, Alexey Pertsev, adria0, Ehud Ben-Reuven, arnaucube, Eduard S., and Marta Bellés. circomlib, 2020. Available at: `https://github.com/tornadocash/circomlib#c372f14d324d57339c88451834bf2824e73bbdbc`.

10 Jordi Baylina, Kobi Gurkan, Roman Semenov, Alexey Pertsev, adria0, Ehud Ben-Reuven, arnaucube, Eduard S., and Marta Bellés. snarkjs, 2020. Available at: `https://github.com/tornadocash/snarkjs#869181cfaf7526fe8972073d31655493a04326d5`.

11 Josh Benaloh and Michael De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology—EUROCRYPT'93: Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23–27, 1993 Proceedings 12*, pages 274–285. Springer, 1994.

12 Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*, pages 561–586. Springer, 2019.

13 Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.6*, 2023.

14 Chainalysis. Understanding tornado cash, its sanctions implications, and key compliance questions, 2022. Available at: `https://blog.chainalysis.com/reports/tornado-cash-sanctions-challenges/`.

15 Dmitry Ermilov, Maxim Panov, and Yury Yanovich. Automatic bitcoin address clustering. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 461–466. IEEE, 2017.

16 Davide Frey, Mathieu Gestin, and Michel Raynal. The synchronization power (consensus number) of access-control objects: The case of allowlist and denylist. *arXiv preprint arXiv:2302.06344*, 2023.

17 Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*, pages 465–482. Springer, 2010.

18 Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security Symposium*, volume 2021, 2021.

19 Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016.

20 Joe Hurd. Verification of the miller–rabin probabilistic primality test. *The Journal of Logic and Algebraic Programming*, 56(1-2):3–21, 2003.

21 Aggelos Kiayias, Markulf Kohlweiss, and Amirreza Sarencheh. Peredi: Privacy-enhanced, regulated and distributed central bank digital currencies. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1739–1752, 2022.

22 Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 839–858. IEEE, 2016.

23 Duc V Le and Arthur Gervais. Amr: Autonomous coin mixer with privacy preserving reward distribution. *ACM Conference on Advances in Financial Technologies (AFT'21)*, 2021.

24 Greg Maxwell. Coinjoin: Bitcoin privacy for the real world. In *Post on Bitcoin forum*, 2013.

25 Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(2):105–121, 2018.

26 Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-knowledge sets. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 80–91. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238183.

27 Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy (SP)*, pages 397–411. IEEE, 2013.

28 Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, et al. An empirical analysis of traceability in the monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 2018(3):143–163, 2018.

29 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Available at: https://bitcoin.org/bitcoin.pdf.

30 Leonid Reyzin and Sophia Yakoubov. Efficient asynchronous accumulators for distributed pki. In *Security and Cryptography for Networks: 10th International Conference, SCN 2016, Amalfi, Italy, August 31–September 2, 2016, Proceedings 10*, pages 292–309. Springer, 2016.

31 Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International workshop on fast software encryption*, pages 371–388. Springer, 2004.

32 Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 459–474. IEEE, 2014.

33 Peter Todd. Merkle mountain ranges, 2018. Available at: https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md.

34 Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. Utt: Decentralized ecash with accountable privacy. Cryptology ePrint Archive, Paper 2022/452, 2022. URL: https://eprint.iacr.org/2022/452.

35 TornadoCash. Tornado.cash governance proposal, 2020. Available at: https://tornado-cash.medium.com/tornado-cash-governance-proposal-a55c5c7d0703.

36 U.S. DEPARTMENT OF THE TREASURY. U.s. treasury sanctions notorious virtual currency mixer tornado cash, 2022. Available at: https://home.treasury.gov/news/press-releases/jy0916.

37 Friedhelm Victor. Address clustering heuristics for ethereum. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*, pages 617–633. Springer, 2020.

38 Zhipeng Wang, Stefanos Chaliasos, Kaihua Qin, Liyi Zhou, Lifeng Gao, Pascal Berrang, Benjamin Livshits, and Arthur Gervais. On how zero-knowledge proof blockchain mixers improve, and worsen user privacy. In *Proceedings of the ACM Web Conference 2023*, pages 2022–2032, 2023.

39 Zhipeng Wang, Marko Cirkovic, Duc V. Le, William Knottenbelt, and Christian Cachin. Pay less for your privacy: Towards cost-effective on-chain mixers. Cryptology ePrint Archive, Paper 2023/1222, 2023. URL: https://eprint.iacr.org/2023/1222.

40 Zhipeng Wang, Xihan Xiong, and William J. Knottenbelt. Blockchain transaction censorship: (in)secure and (in)efficient? Cryptology ePrint Archive, Paper 2023/786, 2023. URL: `https://eprint.iacr.org/2023/786`.

41 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014. URL: `https://ethereum.github.io/yellowpaper/paper.pdf`.

42 Lei Wu, Yufeng Hu, Yajin Zhou, Haoyu Wang, Xiapu Luo, Zhi Wang, Fan Zhang, and Kui Ren. Towards understanding and demystifying bitcoin mixing services. In *Proceedings of the Web Conference 2021*, pages 33–44, 2021.

43 Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. Platypus: a central bank digital currency with unlinkable transactions and privacy-preserving regulation. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2947–2960, 2022.

44 Haaroon Yousaf, George Kappos, and Sarah Meiklejohn. Tracing transactions across cryptocurrency ledgers. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 837–850, 2019.

Non-Atomic Payment Splitting in Channel Networks

Stefan Dziembowski
University of Warsaw, Poland
IDEAS NCBR, Warsaw, Poland

Paweł Kędzior
University of Warsaw, Poland

Abstract

Off-chain channel networks are one of the most promising technologies for dealing with blockchain scalability and delayed finality issues. Parties connected within such networks can send coins to each other without interacting with the blockchain. Moreover, these payments can be "routed" over the network. Thanks to this, even the parties that do not have a channel in common can perform payments between each other with the help of intermediaries.

In this paper, we introduce a new notion that we call *Non-Atomic Payment Splitting (NAPS) protocols* that allow the intermediaries in the network to split the payments recursively into several subpayments in such a way that the payment can be successful "partially" (i.e. not all the requested amount may be transferred). This contrasts with the existing splitting techniques that are "atomic" in that they did not allow such partial payments (we compare the "atomic" and "non-atomic" approaches in the paper). We define NAPS formally and then present a protocol that we call "EthNA", that satisfies this definition. EthNA is based on very simple and efficient cryptographic tools; in particular, it does not use expensive cryptographic primitives. We implement a simple variant of EthNA in Solidity and provide some benchmarks. We also report on some experiments with routing using EthNA.

2012 ACM Subject Classification Security and privacy → Systems security

Keywords and phrases Blockchain, Payment Channels Networks

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.17

Related Version *Full Version*: `https://eprint.iacr.org/2020/166` [7]

Supplementary Material *Software*: `https://github.com/Sam16450/NAPS-EthNA`
archived at `swh:1:dir:5336d35d6eae6b63827c47b1c3614ef501e5083e`

Funding This result is part of a project that received funding from the European Research Council (ERC) under the European Union's Horizon 2020 innovation program (grant PROCONTRA-885666). It was also party financed by the Foundation for Polish Science under grant TEAM/2016-1/4 founded within the UE 2014–2020 Smart Growth Operational Program and by the *Ethereum Foundation* grant FY18-0023.

1 Introduction

Blockchain technology allows a large group of parties to reach a consensus about the contents of an (immutable) ledger, typically containing a list of transactions. In blockchain's initial applications, these transactions described transfers of *coins* between the parties. One of the very promising extensions of the original Bitcoin ledger is blockchains that allow to register and execute the so-called *smart contracts* (or simply "contracts"), i.e., formal agreements between the parties, written down in a programming language and having financial consequences (for more on this topic see, e.g., [12, 6]). Probably the best-known example of such a system is *Ethereum*. Several blockchain-based systems' main limitations are delayed finality, lack of scalability, and non-trivial transaction fees. *Off-chain channels* [18, 3] are a powerful approach for dealing with these issues. The simplest examples of this technology are the so-called "*payment* channels". Informally, such a channel between Alice and Bob is an object in which both parties have some coins that they can freely transfer without interacting with the blockchain ("off-chain"). We explain this in Sec. 1.1 below. Readers familiar with this topic can go quickly over it, just paying attention to some terminology and notation that we use

1.1 Background

Assume that the maximal blockchain reaction time is Δ. We model amounts of coins as non-negative integers and write "n¢" to denote n coins. A payment channel is *opened* when Alice and Bob deploy a smart contract on the ledger and deposit some number of coins (say: x, and y, respectively) into it. The initial *balance* of this channel is: "x¢ in Alice's account, y¢ in Bob's account" (or $[\text{Alice} \mapsto x, \text{Bob} \mapsto y]$ for short). This balance can be *updated* (to some new balance $[\text{Alice} \mapsto x', \text{Bob} \mapsto y']$, such that $x' + y' = x + y$) by exchanging messages between the parties. The corresponding smart contract guarantees that each party can at any time *close* the channel and get the money corresponding to her latest balance. Only the opening and closing operations require interaction with the blockchain. Since updates do not require blockchain participation, each update is immediate (the network speed determines its time) and at essentially no cost.

Now, suppose we are given a set of parties $P_1, \ldots, P_n$ and channels between some of them. These channels naturally form an (undirected) *channel graph*, which is a tuple $\mathcal{G} = (\mathcal{P}, \mathcal{E}, \Gamma)$ with the set of vertices $\mathcal{P}$ equal to $\{P_1, \ldots, P_n\}$ and set $\mathcal{E}$ of edges being a family of two-element subsets of $\mathcal{P}$. The elements of $\mathcal{P}$ will be typically denoted as "$P_i \multimapboth P_j$" (instead of $\{P_i, P_j\}$). Every $P_i \multimapboth P_j$ represents a channel between P_i and P_j, and the *cash function* Γ determines the number of coins available for the parties in every channel. More precisely, every $\Gamma(P_i \multimapboth P_j)$ is a function f of a type $f : \{P_i, P_j\} \to \mathbb{Z}_{\geq 0}$. We will often write $\Gamma^{P_i \multimapboth P_j}$ to denote this function. The value $\Gamma^{P_i \multimapboth P_j}(P)$ denotes the amount of coins that P has in her *account* in channel $P_i \multimapboth P_j$. A *path* (in $\mathcal{G}$) is a sequence $P_{i_1} \multimap \cdots \multimap P_{i_t}$ such that for every j we have $P_{i_j} \multimapboth P_{i_{j+1}} \in \mathcal{E}$. In the formal part of the paper (see Sec. 3.1), we will also include "nonces" in the paths, but in this informal description, we ignore them. In this paper, for the sake of simplicity, we assume that (a) the channel system is deployed with some initial value of Γ, which evolves over time, (b) once a channel system is established, no new channels are created, and no channels are closed (i.e., $\mathcal{E}$ remains fixed), and (c) no coins are added to the existing channels, i.e., the total amount of coins available in every channel $e = P_i \multimapboth P_j$ never exceeds the total amount available in it initially.

Channel graphs can serve for secure payment sending. Let us recall how this works in the most popular payment channel networks, such as *Lightning* or *Raiden*. Our description is very high-level (for the details, see, e.g., [18]). Consider the following example: we have three parties: P_1, P_2, and P_3 and two channels: $P_1 \multimapboth P_2$ and $P_2 \multimapboth P_3$ between them. Now, suppose the *sender* P_1 wants to send v¢ to the *receiver* P_3 over the path $P_1 \multimap P_2 \multimap P_3$, with P_2 being an *intermediary* that *routes* these coins. This is done as follows. First, party P_1 asks P_2 to forward v¢ in the direction of P_3 (we call such a request *pushing* coins from P_1 to P_2). The receipt from P_3 confirming that she received these coins has to be presented by P_2 within 2Δ (denote this receipt with ρ). If P_2 manages to do it by this deadline, she gets these coins in her account in the channel $P_1 \multimapboth P_2$. To guarantee this, P_1 initially blocks these coins in the channel $P_1 \multimapboth P_2$. These coins can be claimed back by P_1 if time 2Δ have passed, and P_2 did not claim them. In a similar way, P_2 pushes these coins to P_3, i.e., she offers P_3 to claim (by providing proof ρ within Δ time) v¢ in the channel $P_3 \circ\!\!-\!\!\circ P_4$. Now suppose that party P_3 claims her v¢ in channel $P_2 \circ\!\!-\!\!\circ P_3$. This can only be done by providing a receipt ρ confirming that she received these coins. We call this process *acknowledging* payment. Party P_2 can now claim her coins in channel $P_1 \circ\!\!-\!\!\circ P_2$ by submitting an acknowledgment containing the receipt ρ. In the above example, the number of coins that can be pushed via a channel $P_i \circ\!\!-\!\!\circ P_{i+1}$ is upper-bounded by the number of coins that P_i has in this channel. Therefore the maximal amount of coins that can be pushed over path $P_1 \to P_2 \to P_3$ is equal to the minimum of these values. We will call this value the *capacity* of a given path.

On the technical level, in the Lightning network, the receipt ρ is constructed using so-called *hash-locked transactions* and "smart contracts" that guarantee that nobody loses money. This is possible thanks to how the $n\Delta$ deadlines in the channels $P_1 \circ\!\!-\!\!\circ P_2$ and $P_2 \circ\!\!-\!\!\circ P_3$ are chosen. An exciting feature of this protocol is that receipt ρ serves not only for internal purposes of the routing algorithm but can also be viewed as the output of the protocol, which can be used by P_1 as a receipt that she transferred some coins to P_4. In other words: P_1 can use ρ to resolve disputes with P_4, either in some smart contract (deployed earlier and using the given PCN for payments) or outside the blockchain. The notion of payment channels can be generalized to "*state* channels". Informally, such channels can serve not only for payments between the parties but also for executing contracts within them. For more on this, see, e.g., [6, 15, 2].

1.2 Our contribution and related work

One of the main problems with the existing PCNs is that sending a payment between two parties requires a path from the sender to the receiver with sufficient capacity. This problem is amplified by the fact that the capacity of potential paths can change dynamically as several payments are executed in parallel. Although usually, the payments are swift, in the worst case, they can be significantly delayed since each "hop" in the network can take as long as the pessimistic blockchain reaction time. Therefore it is hard to predict a given path's capacity even in the very near future. This is especially a problem if the capacity of a given channel is close to being completely exhausted (i.e. it is close to zero because of several ongoing payments). A natural solution for solving this problem is to split the payments into several subpayments. This was described in several recent papers (see, e.g., [16, 17, 20, 8]). However, up to our knowledge, all these papers considered so-called "*atomic* payment splitting", meaning that either all the subpayments got through or none of them. In this paper, we prove a new, alternative technique that we call "*non*-atomic payment splitting" that does not have this feature and hence is more flexible. (We compare atomic and non-atomic splitting in Sec. 2.1.2.) More concretely, our contribution can be summarized as follows.

NAPS definition. We introduce the concept of *non-atomic payment splitting* by defining formally a notion of *Non-Atomic Payment Splitting (NAPS)* protocols. In our definition, we require that splitting is done ad-hoc by the intermediaries, possibly in reaction to dynamically changing the capacity of the paths or fees. Perhaps the easiest way to describe NAPS is to look at payment networks as tools for outsourcing payment delivery. For example, in the scenario from Section 1.1 party P_1 outsources to P_2 the task of delivering v¢ to P_4, and gives P_2 time 2Δ to complete it (then P_2 outsources this task to P_3 with a more restrictive deadline). The sender might not be interested in *how* this money is transferred, and the only thing that matters to her is that it is indeed delivered to the receiver and that she gets the receipt. In particular, the sender may not care if the money gets split on the way to the receiver, i.e., if the coins that he sends are divided into smaller amounts that are transferred independently over different paths. In many cases, the sender may also be ok with not all money being transferred at once. NAPS protocol permits such recursive non-atomic payment splitting into "subpayments" and partial transfers of coins. This splitting can be done in an ad-hoc way. Moreover, the users can try to route the same payment over the same path multiple times (hoping that some more capacity becomes available in the meantime). We present a UC-like definition of NAPS. An additional advantage of our contribution is that our definition can be easily adapted to cover the *atomic* payment splitting protocols [17, 1, 19, 8].

EthNA construction. We construct a protocol that we call EthNA that satisfies the NAPS definition. We call our protocol EthNA, in reference to Etna, one of the highest active volcanos in Europe. This is because the coin transfers in EthNA resemble a lava flood (with large streams recursively bifurcating into small substreams). The letter "h" is added so that the prefix "Eth-" is reminiscent of ETH, the symbol of Ether (the currency used in Ethereum), and "NA" stands for "Non-Atomic".

In EthNA the "subreceipts" for subpayments are aggregated by the intermediaries into one short subreceipt so that their size does not grow with the number of aggregated subreceipts. This is done efficiently, particularly by avoiding advanced and expensive techniques such as noninteractive zero knowledge or homomorphic signature schemes and hash functions. Instead, we rely on a technique called "fraud-proofs" in which an honest behavior of parties is enforced by a punishing mechanism (this method was used before, e.g., in [21, 6]). We stress that the amount of data that is passed between two consecutive parties on the path does *not* depend on the number of subpayments in which the payment is later divided. The same applies to the data these two parties send to the blockchain if they conflict. We summarize the complexity of EthNA in Sec. 3.3.

Security analysis and implementation. We provide a formal security analysis of EthNA. More precisely, we prove that EthNA satisfies the NAPS definition. We also analyze EthNA's complexity. We also implement EthNA contracts in Solidity (the standard language for programming the smart contracts in Ethereum), and we provide some routing experiments. We describe this implementation and provide some benchmarks. We stress, however, that routing algorithms are *not* the main focus of this work and further research on designing algorithms that exploit the non-atomicity of payment splitting.

Possible applications of NAPS. As mentioned above, one obvious application of NAPS is to help efficiently send one big payment by dividing it into several ad-hoc installments: if it is impossible to route the total amount u, then the client can accept the fact that $v < u$ coins were transferred (due to network capacity limitations), and try to transfer the remaining $u - v$ coins later (in another installment). The same applies to other situations, e.g., when the user wants to exchange coins for another currency. Ideally, he would like to exchange the entire amount u, but exchanging $v < u$ is better than nothing. A related scenario is making a partial "bank deposit" when the user wants to deposit as much money as possible but no more than u.

Moreover, in many cases, the goods the seller delivers in exchange for the payment can be divided into tiny units and sent to the buyer depending on how many coins have been transferred. One example is battery charging, where charging, say 1/2 of the battery is much better than having the battery dead. This applies both to mobile phones and to IoT devices that can trade energy with each other. Let us also mention applications like file sharing, where the client typically connects to several servers and tries to download as much data as possible from each. NAPS can be an attractive way to perform payments in this scenario. Note also that NAPS can be combined with other means of payment. If a user manages to send only $u < v$ coins via NAPS, then she can decide to send the rest $(v - u)$ in some more expensive way (this makes sense, especially in systems where the fee depends on the amount being transferred, e.g., in the credit card payments).

Related work. Some of the related work was mentioned already before. Off-chain channels are a topic of intensive research, and there is no space here to describe all recent exciting developments [9, 15, 13, 6, 4, 5, 11, 14, 2] in this area. The reader can also consult SoK papers on off-chain techniques (e.g. [10]). Partial coin transfers were considered in [17], but with no aggregation techniques and ad-hoc splitting. Atomic payment splitting has been considered in [17, 1, 19, 8]. All of these papers focus on routing techniques, which is not the main topic of this paper.

Organization and notation. Sec. 2 contains an informal description of our ideas. Then, in Sec. 3, we provide the formal NAPS definition and the detailed description of EthNA and its security properties. An overview of our implementation and simulations is presented in Sec. 3.3. When we say that a message is "signed by some party", we mean that it is signed using some fixed signature scheme that is existentially unforgeable under a chosen-message attack. Natural numbers are denoted with $\mathbb{N}$. We will also use the notion of *nonces*. Their set is denoted with $\mathcal{N}$. We assume that $\mathcal{N} = \mathbb{N}$. We use some standard notations for functions, string operations, and trees. By $[a_i \mapsto x_1, \ldots, a_m \mapsto x_m]$ we mean a function $f : \{a_i, \ldots, a_m\} \to \{x_1, \ldots, x_m\}$ such that for every i we have $f(a_i) := x_i$. Let A be some finite alphabet. Strings $\delta \in A^*$ are frequently denoted using angle brackets: $\delta = \langle \delta_1, \ldots, \delta_m \rangle$. Let δ be a string $\langle \delta_1, \ldots, \delta_n \rangle$. For $i = 1, \ldots, n$ let $\delta[i]$ denote δ_i. Let ε denote an empty string, and "$||$" denote the concatenation of strings. We overload this symbol, and write $\delta||a$ and $a||\delta$ to denote $\delta||\langle a \rangle$ and $\langle a \rangle||\delta$, respectively (for $\delta \in A^*$ and $a \in A$). For $k \leq n$ let $\delta|_k$ denote δ's prefix of length k. A set of prefixes of δ is denoted $\mathsf{prefix}(\delta)$ (note that it includes ε).

We define trees as prefix-closed sets of words over some alphabet A. Formally, a *tree* is a subset T of A^* such that for every $\delta \in T$ we have that any prefix of δ is also in T. Any element of T is called a *node* of this tree. For two nodes $\delta, \beta \in T$ such that $\beta = \delta||a$ (for some a) we say that δ is the *parent* of β, and β is a *child* of δ. A *labeled tree over* A is a pair $(T, \mathcal{L})$, where T is a tree over A, and $\mathcal{L}$ is a function from T to some set of *labels*. For $\delta \in T$ we say that $\mathcal{L}(\delta)$ is the *label of* δ.

2 Informal description

Below, in Sec. 2.1 we provide an overview of NAPS definition, and in Sec. 2.2 we informally describe EthNA.

2.1 Overview of the NAPS definition

Let us now explain the NAPS protocol features informally (for a formal definition, see Sec. 3.1). Throughout this paper, we use the following convention: our protocols are run by a set of *parties* denoted $\mathcal{P} = \{P_1, \ldots, P_n\}$, where P_1 be the *sender*, $P_2, \ldots, P_{n-1}$ be the *intermediaries*, and P_n is the *receiver*. A message m signed by a party P_i will be denoted $\wr m \int_{P_i}$. Let v be the number of coins that P_1 wants to send to P_n, and let t be the maximal time until the transfer of coins should be completed. Since, in general, P_1 can perform multiple payments to P_n, we assume that each payment comes with a nonce $\mu \in \mathcal{N}$ that can be later used to identify this payment. Sometimes we will simply call it "payment μ". For simplicity, we start with an informal description of how NAPS protocols operate when all parties are honest. The security properties (taking into account the malicious behavior of the parties) are described informally in Sec. 2.1.1, and formally defined in Sec. 3.1. Before proceeding with the description of EthNA the reader may look at the example in Fig. 1.

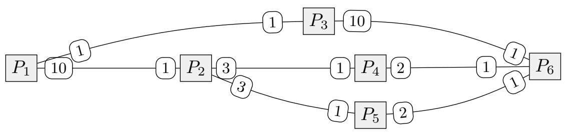
To describe the protocol more generally, let us start by presenting it from the point of view of the sender P_1. Let $P_{i_1}, \ldots, P_{i_t}$ be the neighbors of P_1, i.e., parties with which P_1 has channels. Suppose the balance of each channel $P_1 \multimapboth P_{i_j}$ is $[P_1 \mapsto x_i, P_{i_j} \mapsto y_j]$ (meaning that P_1 and P_{i_j} have x_i and y_j coins in their respective accounts in this channel). Now, P_1 chooses to push some amount v_j of coins to P_n via some P_{i_j}, and set up a deadline t_j for this (we will also call v_j a *subpayment* of payment μ). This results in: (a) balance $[P_1 \mapsto x_i, P_{i_j} \mapsto y_j]$ changing to $[P_1 \mapsto x_i - v_j, P_{i_j} \mapsto y_j]$, (b) the number of coins that P_1 still wants to transfer to P_n is decreased as follows: $v := v - v_j$, and (c) P_{i_j} holding "v_j coins that she should transfer to P_n within time t_j.

It is also ok if P_{i_j} transfers only some part $v'_j < v_j$ of this amount (this can happen, e.g., if the paths that lead to P_n via P_j do not have sufficient capacity). In this case, P_1 has to be given back the remaining ("non-transferred") amount $r = v_j - v'_j$. More precisely, before time t_j comes, party P_{i_j} acknowledges the amount v'_j that she managed to transfer. This results in (1) changing the balance of the channel $P_1 \multimapboth P_{i_j}$ by crediting v'_j coins to P_{i_j}'s account in it, and (2) r coins to P_1's account. Moreover, (3) P_1 adds back the non-transferred amount r to v, by letting $v := v + r$. Above (1) corresponds to the fact that P_{i_j} has to be given the coins that she transferred, and (2) comes from the fact that not all the coins were transferred (if P_{i_j} managed to transfer all the coins, then, of course, $r = 0$). Finally, (3) is used for P_1's "internal bookkeeping" purposes, i.e., P_1 simply writes down that r coins "were returned" and still need to be transferred. While the party P_1 waits for P_{i_j} to complete the transfer that it requested, she can also contact some other neighbor P_{i_k} asking her to transfer some other amount v_k to P_n. This is done in the same way as transferring coins via P_{i_j}.

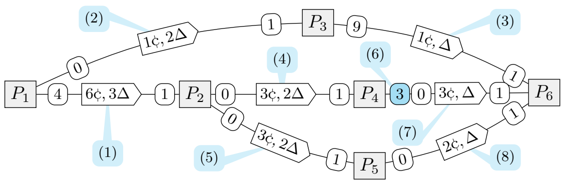
The intermediaries can repeat this process. Let P be a party that holds some coins that were "pushed" to her by some P' (which originate from P_1 and must be delivered to P_n). Now, P can split them further, and moreover, she can decide on her own how this splitting is done depending, e.g., on the current capacity of the possible paths leading to P_n. The payment splitting can be done arbitrarily, except for the two following restrictions. First, we do not allow "loops" (i.e. paths containing the same party more than once), as it is hard to imagine any application of such a feature. In the basic version of the protocol, we assume that the number of times a given payment subpayment is split by a single party P is bounded by a parameter $\delta \in \mathbb{N}$, called *arity* (for example arity on Figs. 1 is at most 2). In the extended version of this paper [7] we present an improved protocol where δ is unbounded (at the cost of a mild increase of the pessimistic number of rounds of interaction). As mentioned, the essential feature of NAPS is the *non-atomicity of payments*. We discuss it further below.

2.1.1 NAPS security properties

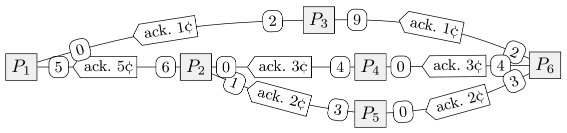
In the description in Sec. 2.1 we assumed that all parties behaved honestly. Like all other PCNs, we require that NAPS protocols work if the parties are malicious. In particular, no honest party P can lose money, even if all the other parties are not following the protocol and are working against P. The corrupt parties can act in a coalition modeled by an adversary $\mathcal{A}$. Formal security definition appears in Sec. 3.1. Let us now informally list the security



(a) The channel graph with the initial coin distribution.



(b) The sender P_1 wants to send 7¢ to the receiver P_6. She splits these coins into two amounts: 6¢ pushed to P_2 and 1¢ pushed to P_3. This is indicated with labels (1) and (2), respectively. Then (3) party P_3 simply pushes 1¢ further to P_6. Party P_2 splits 6¢ into 3¢ + 3¢, and pushes 3¢ to both P_4 (4) and P_5 (5). Path $P_4 \rightarrow P_6$ initially had capacity 2 only (see Fig. (a) above), but luckily in the meanwhile 1¢ got unlocked (6) for P_4 in channel $P_4 \circ\!\!-\!\!\circ P_6$, and hence (7) party P_4 pushes all 3¢ to P_6. Party P_5 pushes only 2¢ to P_6 (8). The channel balances correspond to the situation *after* the coins are pushed (except of channel $P_4 \circ\!\!-\!\!\circ P_6$ where we also indicated the fact that 1¢ got unlocked (6)). Each party P can also decide on her own about the timeout t of each subpayment she pushes (this timeout is indicated with "$x\Delta$"). The only restriction is that t has to come at least Δ before the time she has to acknowledge that subpayment back. This is because P needs this "safety margin" of Δ in case P' is malicious, and the acknowledgment has to be done "via the blockchain".



(c) Party P_6 acknowledges subpayment of 1¢ to P_3, which, in turn acknowledges it to P_1. Party P_6 also acknowledges subpayment of 3¢ to P_4 and 2¢ to P_5, who later acknowledge them to P_2. Once P_2 receives both acknowledgments, she "aggregates" them into a single acknowledgment (for 5¢) and sends it to P_1. As a result 5¢ + 1¢ = 6¢ are transferred from P_1 to P_6. The channel balances correspond to the situation *after* the coins were acknowledged.

Figure 1 An example of a NAPS protocol execution. An edge "$\boxed{P_i}\!-\!(x)\!-\!\!-\!\!-\!(y)\!-\!\boxed{P_j}$" denotes the fact that there exists a channel between P_i and P_j, and the parties have x and y¢ in it, respectively.

requirements, which are pretty standard and hold for most PCNs (including Lightning). Below, let u denote the total amount of coins P_1 wants to transfer to P_n within some payment μ.

The first property is called *fairness for the sender.* To define it, note that as a result of payment μ (with timeout t), the total amount of coins that each party P has in the channels with other parties typically changes. Let $net_\mu(P)$ denote the number of coins that P gained in all channels. Of course $net_\mu(P)$ can be negative if P lost $-net_\mu(P)$ coins. We require that by the time t an honest P_i holds a receipt of a form $\mathsf{Receipt}(\mu, v)$:="*an amount v of coins has been transferred from P_1 to P_n as a result of payment μ*". Moreover, under normal circumstances, i.e. when everybody is honest, v is equal to $-net_\mu(P_1)$ (i.e. the sum of the amounts that P_1 lost in the channels). In case some parties (other than P_1) are dishonest, the only thing that they can do is to behave irrationally and let $v \geq -net_\mu(P_1)$, in which case P_1 holds a receipt for transferring *more* coins than she actually lost in the channels. A receipt can be later used in another smart contract (e.g., a contract that delivers some digital goods whose amount depends on v). *Fairness for the receiver* is defined analogously, i.e.: if P_1 holds a receipt $\mathsf{Receipt}(\mu, v)$ then typically $v = net(P_n)$, and if some parties (other than P_n) are dishonest, then they can make $v \leq net_\mu(P_n)$. In other words, P_1 cannot get a receipt for an amount higher than what P_n actually received in the channels. Finally, we require that the following property called *balance neutrality for the intermediaries* holds: for every honest $P \in \{P_2, \ldots, P_{n-1}\}$ we have that $net_\mu(P) \geq 0$. Again: if everybody else is also honest, then we have equality instead of inequality.

2.1.2 Atomic vs. non-atomic payment splitting

As already highlighted in Sec. 1.2, the previous protocols on payment splitting always required payments to be atomic, meaning that for a payment to succeed, all the subpayments had to reach the receiver. Technically, this means that to issue a receipt for *any* of the subpayments (this receipt is typically a preimage of a hash function, see, e.g., [8]) all of them need to reach the receiver. This has several disadvantages: (1) the coins remain blocked in every path at least until the last subpayment arrives to the receiver, (2) the success of a given subpayment depends not only on the subsequent intermediaries but also on the other "sibling" paths (this problem was observed in [8] where it is argued that this risk may lead to intermediaries rejecting subpayments that were split before, see Sec. 3.1 of [8]). Finally, atomic payments may result in "deadlock" situations in the network where two competing payments can prevent each other from being executed. We describe an example of such a situation in [7].

Let us also remark that "atomicity" and even "fine-grained atomicity" can also be obtained in ETHNA by a small protocol modification. We write more about it in the extended version of this paper [7]. Let us also remark that atomic payment splitting, in general, seems to be easier to achieve, which is probably the reason why there has been more focus on them in the literature (with papers focusing more on other aspects of this problem, such as routing algorithms, e.g. [8]). Finally, let us stress that we do not claim that non-atomicity is superior to atomicity. We think both solutions have advantages and disadvantages, and there exist applications where each is better than the other.

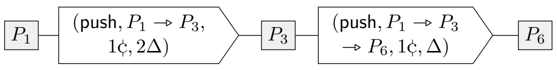
2.2 Overview of the EthNA protocol

After presenting the NAPS definition, let us now explain the main ideas behind the ETHNA protocol that realizes it. An essential feature of ETHNA is that it permits "subreceipt aggregation", by which we mean the following. Consider some payment μ. Once P_n receives some subpayment v that reached it via some path $\pi = P_1 \rightarrow P_{i_1} \rightarrow \cdots P_{i_k} \rightarrow P_n$ she issues a *subreceipt* for this payment and sends it to P_{i_k}. Each intermediary that receives more than one subreceipt can aggregate them into one short subreceipt that she sends further in the direction of P_1. Finally, P_1 also produces one short receipt for the entire payment. This results in small communication complexity, and in particular, the pessimistic gas costs are low (we discuss this in more detail in Sec. 4. One option would be to let the subreceipt be signed using a homomorphic signature scheme and then exploit this homomorphism to aggregate the subreceipts. This paper uses a simpler solution that can be efficiently and easily implemented in the current smart contract platforms.

Very informally speaking, we ask P_n to perform the "subpayment aggregation herself" (this is done when signing a subreceipt and does not require any further interaction with P_n). Then, we let the other parties verify that this aggregation was performed correctly. If any "cheating by P_n" is detected (i.e. some party discovers that P_n did not behave honestly), then proof of this fact (called a "fraud-proof") will count as a receipt that a total amount has been transferred to P_n. From the security point of view, this is ok since an honest P_n will never cheat (hence, no fraud-proof against him will ever be produced). Thanks to this approach, we avoid entirely using any expensive advanced cryptographic techniques (such as homomorphic signatures or noninteractive proofs). Below we explain the main idea of EthNA by considering the example from Fig. 1. Again, we start by describing how the protocol works when everybody is honest, and then (in Sec. 2.2) we show how the malicious behavior is prevented.

Invoice sending. The protocol starts with the receiver P_n sending to P_1 an "invoice" that specifies (among other things) the identifier μ of the payment, and the maximal amount u of coins that P_n is willing to accept. As we explain below, this invoice may be later used together with fraud-proofs to produce proof that all u coins were transferred to P_n (if she turns out to be malicious).

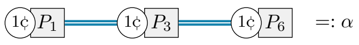
Pushing subpayments. Pushing subpayments is done by sending messages containing information about the path that the subpayment "traveled" so far (together with the number of coins to be pushed and timeout information) and simultaneously blocking coins in the underlying channels. The messages sent between P_1, P_3 and P_6 in Fig. 1 (a)) are presented in the picture below.


P_1
$(\mathsf{push}, P_1 \rightarrow P_3, 1¢, 2\Delta)$
P_3
$(\mathsf{push}, P_1 \rightarrow P_3 \rightarrow P_6, 1¢, \Delta)$
P_6

Whenever a message $(\mathsf{push}, \pi, v, t)$ is sent from P to P', the party P blocks v coins in channel $P \circ\!\!-\!\!\circ P'$ for time t. These coins are claimed by P' if she provides a corresponding subreceipt within time t. Otherwise, they are claimed back by P.

Acknowledging subpayments by the receiver. The receiver P_n acknowledges the subpayments by replying with a signed subreceipt and claiming the coins blocked in the corresponding channels. At the same time, the receiver P_n constructs a labeled graph called the "payment tree" that is stored locally by P_n and grows with each acknowledged subpayment.

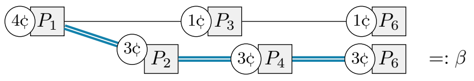
Let us now explain how the payment tree is constructed. Consider again Fig. 1 (c). As explained before, the order of message acknowledgment can be arbitrary. In what follows, we assume that the receiver P_6 first acknowledges the subpayment that came along the path $P_1 \rightarrow P_3 \rightarrow P_6$. This means that P_6 "accepts" that 1¢ will be transferred to her from P_1 via path $P_1 \rightarrow P_3 \rightarrow P_6$, or, in other words: 1¢ will be "passed" through each of P_1, P_3, and P_6 (note that we included here the sender P_1 and the receiver P_6). This can be depicted as the following graph that consists of a single path that we denote α:



(1)

To acknowledge the subpayment that was pushed along the path $P_1 \rightarrow P_3 \rightarrow P_6$ party P_6 signs α and sends it to P_3. Such signed information will be called a "subreceipt" and denoted $\wr\alpha\int_{P_n}$. By providing this subreceipt, party P_6 also gets 1¢ in the $P_3 \circ\!-\!\circ P_6$ (these coins were blocked by P_3 in this channel when the "push" message was sent). The graph from Eq. (1) is the first version of the payment tree that, as mentioned above, the receiver P_6 stores locally.

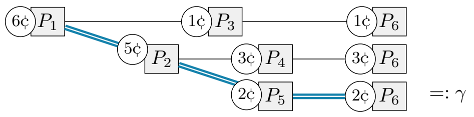
Now, suppose the next subpayment that P_6 wants to acknowledge is the one that came along the path $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_6$, i.e., P_6 accepts that 3¢ will be transferred to her from P_1 via path $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_6$. The receiver P_6 now modifies the payment tree as follows:



(2)

Analogously to what we saw before, this tree represents the total amount of coins that will be "passed" through different parties from P_1 to P_6 after acknowledging this subpayment is completed. In Eq. (2) the thick line (denoted β) corresponds to the "new" path, and the thin one is taken from Eq. (1), except that P_1 is labeled with "4¢". This is because the total amount of coins that will be passed through P_1 is equal to the sum of the coins passed before (1¢) and now (3¢). Party P_6 now signs path β to create a subreceipt that she sends to P_4 to claim 3¢ in the channel $P_4 \circ\!-\!\circ P_6$.

Finally, P_6 acknowledges the subpayment along the path $P_1 \rightarrow P_2 \rightarrow P_5 \rightarrow P_6$. This is done similarly to what we did before. The resulting tree is now as follows.



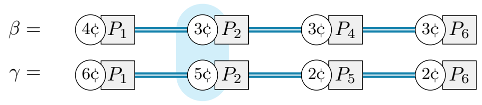
(3)

Note that we performed "summing" in two places on Eq. (3): at the node P_1 (where we computed 6¢ as 4¢ + 2¢) and an P_2 (where 5¢ = 2¢ + 3¢). Labeled path γ is now signed by P_6 and sent to P_5 as subreceipt in order to claim 2¢.

The payment trees whose examples we saw in Eqs. (1)–(3) are defined formally (in a slightly more general version) in Sec. 3.2. Their main feature is that the value of coins in the label of each node P is equal to the sum of the labels of the children of P. A recursive application of this observation implies that the coin value of a label of P is equal to the sum of labels in the leaves of the subtree rooted in P. In particular: the label on the root of the entire tree equals the sum of the values in the leaves.

Acknowledging subpayments by the intermediaries. We now show how the intermediaries $P_2, \ldots, P_{n-1}$ acknowledge the subpayments. On a high level, this is done by propagating the subreceipts (issued by P_n) from right to left. Each party may receive several such subreceipts (if she decides to split a given subpayment). Let $\mathcal{W}$ be the set of such subreceipts . When a party P wants to acknowledge the subpayment, she chooses (in a way that we explain below) one of the subreceipts ζ from her set $\mathcal{W}$. She then forwards it back in the left direction to the party P' that pushed the given subpayment to her. As a result P gets v¢ in the channel $P' \circ\!-\!\circ P$. To determine the value of v¢ the following rule is used: it is defined as the label of P on the path ζ. Given this, the rule for choosing $\zeta \in \mathcal{W}$ is pretty natural: P simply chooses such the ζ that maximizes v. Such ζ will be called a "leader" of $\mathcal{W}$ (at node P). To illustrate it, let us look again at our example from Fig. 1.

First, observe that P_3 holds only one subreceipt (i.e., the signed path α). She simply forwards it to P_1 and receives 1¢ in the channel $P_1 \multimapboth P_3$. Note that this is exactly equal to the value that she "lost" in the channel $P_3 \multimapboth P_6$, and hence the balance neutrality property holds. The situation is a bit more complicated for P_2 since she holds two paths signed by the receiver: β (defined on Eq. (2)) and γ (from Eq. (3)). By applying the rule described above, P_2 chooses the leader ζ at P_2 to be equal to γ (since 5¢ > 3¢). This is depicted below (the shaded area indicates the labels that are compared).



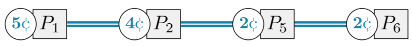
(4)

What remains is to argue about balance neutrality for P_2, i.e., that number of coins received by P_2 in the channel $P_1 \multimapboth P_2$ is equal to the sum of coins that she "lost on the right-hand side". In this particular example, it can be easily verified just by looking at Eq. (4) (5¢ are "gained", and 2¢ + 3¢ are "lost"). In the general case, the formal proof is based on the property that the value of coins in the label of each node P in a payment tree is equal to the sum of the labels of the children of P.

Final receipt produced by P_1. Once all subpayments are completed, P_1 decides to conclude the procedure and obtain the final receipt for the entire payment (see Sec. 2.1.1). Again, P_1 holds a "payment report" $\mathcal{W}$, i.e. a set of paths signed by P_6. In the case of our example, these paths are α (sent to P_1 by P_3) and γ (sent by P_2). Party P_1 chooses her "receipt" similarly as the intermediaries choose which subreceipt to forward. More precisely, let ζ be the path that is the leader of $\mathcal{W}$ at node P_1. This path becomes the final receipt. The amount of transferred coins equals the label of P_1 in ζ. In our case, the leader ζ is clearly γ (since its label at P is "6¢", while the label of γ at P is "1¢", cf. Eqs (1) and (3)). Hence, γ becomes the final receipt for the payment of 6 coins.

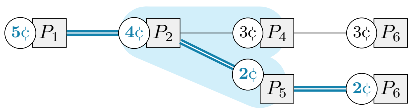
"Fairness for the sender" follows the same argument as "balance neutrality for the intermediaries". For "fairness for the receiver," observe that ζ is signed by the receiver and is taken from the payment tree (created and maintained by the receiver). To finish the argument, recall that: (a) as observed before, the label in the root of such a tree is always equal to the sum of the labels in its leaves, and (b) this sum is exactly equal to the total amount of coins that the receiver received from its neighbors during this payment procedure.

Dealing with malicious behavior. The primary type of malicious behavior that we have to deal with is cheating by the receiver P_n, whose goal could be to get more coins than appears on the final receipt held by the sender P_1. This could potentially be done at the cost of P_1 or some of the intermediaries. So far, we have not described how to guarantee that P_n produces the subreceipts correctly. As already highlighted, our trick is to let a malicious P_n arbitrarily produce the subreceipts and later let other parties verify P_n's operation. This is based on the idea of fraud-proofs: if an intermediary P finds proof that P_n is cheating, she can automatically claim all coins that were pushed to her by forwarding this proof "to the left". In this way, the cheating proof reaches the sender P_1, who can now use it as the receipt for transferring the total amount that was requested (recall that P_1 holds an "invoice" from P_n). Suppose, e.g., that in our scenario P_6 cheats by sending to P_5, instead of γ (see Eq. (3)), the following subreceipt:

$\widehat{\gamma} :=$ 

(5)

The receiver does it to make P_1 hold a receipt for 5¢, while in fact receiving 6¢. Party P_5 has no way to discover this fraud attempt (since from her local perspective everything looks ok), so 2¢ get transferred to P_6 in the channel $P_5 \multimap P_6$. Party P_5 forwards $\widehat{\gamma}$ to P_2 and gets 2¢ in the channel $P_2 \multimap P_5$ (hence the "balance neutrality" property for her holds). Now look at this situation from the point of view of P_2. In addition to $\widehat{\gamma}$ she got one more subreceipt, namely β (see, e.g., Eq. (4)). Party P_2 preforms a "consistency check" by combining $\widehat{\gamma}$ and β. This is done by trying to locally reconstruct the part of the payment tree that concerns P_2. This is done as follows. First observe that the value on the label of P_1 in β is 4¢, which is smaller than the label of P_1 in $\widehat{\gamma}$ (which is equal to 5¢). This means that β had to be signed by P_6 *before* she signed $\widehat{\gamma}$. Hence P_2 first writes down β, and then on top of it she writes $\widehat{\gamma}$ (possibly overwriting some values). Normally (i.e. when P_6 is honest), this should result in a subtree of the tree from Eq. (3). However, since P_6 was cheating the resulting graph is different. Namely, P_2 reconstructs the following:



(6)

It is now obvious that P_6 is cheating since the labels on the children of P_2 sum up to 5¢, which is larger than 4¢ (the label of P_2). This "inconsistency" is marked as a shaded region on Eq. (6). Hence the set $\{\beta, \widehat{\gamma}\}$ is a fraud-proof against P_6. As described above, once we get such proof, we are "done": each intermediary can claim all money that was blocked for her, and the receiver can use it as a receipt that *all* the coins were transferred. Let us stress that, of course, none of the parties assumes a priori that P_6 is honest, and hence the "consistency check" is always performed.

3 Technical details

We now proceed to the formal exposition of the ideas already presented informally in Sec. 2. We start with defining a generalization of the term "paths" that were informally introduced before. As already explained, to be as general as possible, the NAPS definition permits that several subpayment of the same payment μ are routed via the same party independently. Consider, e.g., the following scenario: 2¢ is sent from P_1 to P_4 via a path $P_1 \multimap P_2 \multimap P_3 \multimap P_4$. This amount is first split by P_2 as: 1¢ + 1¢, and each 1¢ coin is pushed to P_3, who, in turn, pushes each of them further to P_4. Obviously, both 1¢ coins traveled along $P_1 \multimap P_2 \multimap P_3 \multimap P_4$, but nevertheless, they have to be considered as separate subpayments. In order to uniquely identify each of them, we augment the definition of "path" to include also "nonces" that will make them unique (in the abovementioned situations). To distinguish such paths from those we used in the informal part we denoted them as strings of pairs (party,nonce). A nonce is added in every hop. For example, in the above scenario: the (augmented) paths are as follows $\langle (P_1, \mu_1), (P_2, \mu_2), (P_3, \mu_3), (P_4, \mu_4) \rangle$ and $\langle (P_1, \mu_1), (P_2, \mu_2), (P_3, \mu_3'), (P_4, \mu_4') \rangle$ (where for both $i = 3, 4$ we have that μ_i and μ_i' are distinct). Moreover, we assume that μ_1 ("contributed" by the sender P_1) is equal to the nonce that identifies the entire payment.

Formally, for a channel graph $\mathcal{G} = (\mathcal{P}, E, \Gamma)$ a string $\pi = \langle (P_{i_1}, \mu_1), \ldots, (P_{i_{|\pi|}}, \mu_{|\pi|}) \rangle$ is a *path over $\mathcal{G}$ (for payment μ)* if each $\mu_i \in \mathcal{N}$ is a nonce, each $P_{i_j} \multimapboth P_{i_{j+1}}$ is an edge in $\mathcal{G}$, and $P_{i_1} = P_1$. We also assume that a path corresponding to a payment μ always starts with (P_1, μ). We say that P *appears on π (at position j)* if we have that $P = P_{i_j}$. We assume that every P appears at most once on π, or, in other words: the paths have no loops. In the sequel, every *party* or *functionality* is modeled as poly-time interactive Turing machine. Throughout this section, P denotes a party, u, v and w are non-negative integers denoting the amounts of coins, μ is a nonce, π is a path over $\mathcal{G}$, and t is time.

3.1 NAPS formal security definition

The protocol is parameterized with a security parameter 1^κ known to all machines. The protocol is executed by parties $P_1, \ldots, P_n$, who know each other's public keys (this is easy to achieve in real life using existing underlying blockchain infrastructure). The protocol also comes with an incorruptible party *RVM* called *receipt verification machine*. The role of this machine is to verify a receipt issued by P_n for payment μ. If this machine outputs $(\textsf{i-sent}, \mu, w)$ to $\mathcal{Z}$ then we consider payment μ to be completed with the total amount of w coins transferred from P_1 to P_n. It models that the receipts produced by P_1 need to be publicly verifiable, so, e.g., they can be used later in another smart contract, see Sec. 2.1.1. Following the tradition in formal cryptography, we first describe how network communication is organized. Then we introduce the notions of "adversary" and "environment". Afterward, we specify the security requirements of the protocol by describing the "ideal" and "real" models. Finally, we define security by comparing these two models. Both the ideal and the real model come with a functionality $\textsf{Accounts}_{\mathcal{G}}$. This functionality (for the lack of space, presented in detail in the extended version of this paper [7]) is used to model the amounts of coins that the parties have in the channels. It is initialized with $\mathcal{G}$ and accepts messages $(\textsf{trans}, P_i, P_j, v)$ that are used to transfer v coins from P_i to P_j in channel $P_i \multimapboth P_j$.

The network model. We assume a synchronous communication network, i.e., the execution of the protocol happens in rounds. The notion of rounds is just an abstraction that simplifies our model and has been used frequently in this area in the past (see, e.g., [5, 6]). Whenever we say that some operation (e.g. sending a message or simply staying in idle state) *takes between τ and τ' rounds*, we mean that it is up to the adversary to decide how long this operation takes (as long as it takes between τ and τ' rounds). The same convention applies to statements like "it takes at most/at least τ rounds". We assume that every machine is activated in each round. The communication between every two parties P and P' and between a party and an ideal functionality takes 1 round. The adversary can delay messages sent between other machines by at most Δ rounds. This will always be stated explicitly. The links between all the entities in the system are secure (encrypted and authenticated). To avoid replay attacks, we assume that every party (both in the ideal and real scenario) rejects a message m if she already received m before. Messages are tuples starting with keywords written in sans-serif. We also use the following convention. When we say that a party waits to receive a "message m of a form F", we mean that all messages of a different form are ignored. For example, if form F is $(\textsf{i-push}, (\pi || \langle (P, \mu), (P', \mu') \rangle, v, t)$ this means that m has to start with an "i-push" keyword, followed by a parameter denoting a path that ends with two elements (denoted (P, μ) and (P', μ') for future reference), parameter v denoting a number of coins, and t denoting time.

The adversary and the environment. The protocol is attacked by a poly-time rushing adversary $\mathcal{A}$ who can *corrupt* some parties (when a party is corrupt $\mathcal{A}$ learns all its secrets and takes complete control over it). A party that has not been corrupt is called *honest.* To model that honest parties can make internal decisions about the protocol actions, we use the concept of an *environment.* This notion is taken from the UC framework; however, to keep things simple, we do not provide a complete UC-composable analysis of our protocol (in particular: since we do not aim at proving composability, we do not have the "session ids," and we use a simplified modeling of time). The environment and the adversary take as input $\mathcal{G}$ and the security parameter. The environment and the adversary can freely read the state of the $\mathsf{Accounts}_{\mathcal{G}}$ functionality. Additionally, we allow the ideal-model adversary to transfer coins from a dishonest party to an honest one. This corresponds to the fact that we allow corrupt parties to behave irrationally and lose coins. $\mathcal{A}$ and $\mathcal{Z}$ can communicate. At the end of its execution, $\mathcal{Z}$ produces an output.

The ideal model. Following the conventions of the UC framework, we assume that in the ideal model, the parties simply forward to the ideal functionality the messages that they receive from $\mathcal{Z}$. For a NAPS protocol with arity δ executed over graph $\mathcal{G}$ the corresponding ideal functionality is denoted $\mathsf{NAPS}^{\delta}_{\mathcal{G}}$ and presented on Fig. 2.

The messages exchanged in the ideal model are indicated with a prefix "i-". Let us now discuss the messages exchanged between the parties and the ideal functionality parties . Note that this functionality does not explicitly send any messages to the simulator. The simulator interacts with the ideal functionality via the corrupt parties she controls. To initiate a new payment μ parties P_1 and P_n send respectively a message $\mathsf{i\text{-}send}(\mu, v, t)$ to P_1 and $\mathsf{i\text{-}receive}(\mu, v, t)$ to P_n. We require that these messages have to be sent simultaneously by P_1 and P_n. This corresponds to an assumption that the parties P_1 and P_n agreed on transferring the coins beforehand. Once the transfer is finished, party *RVM* receives a message $\mathsf{i\text{-}acknowledged}(\langle(P, \mu)\rangle, s)$ from the ideal functionality. The functionality $\mathsf{NAPS}^{\delta}_{\mathcal{G}}$ maintains a set Ψ that contains all the push requests that have not yet been acknowledged. By push *requests* we mean tuples (π, v, t) such that some party sent $(\mathsf{i\text{-}push}, \pi, v, t)$ to the functionality. If such a push request is in Ψ then we say it is *open.* This indicates that the functionality is currently working on pushing v coins that already "traveled" along the path π, and the deadline for this is t. The amount of coins still waiting to be delivered is maintained using the function *remaining.* The push requests are created recursively. Suppose there is an open push request $(\pi || \langle(P, \mu)\rangle, v, t)$. To push it to a party P' party P sends a message $(\mathsf{i\text{-}push}, (\pi || \langle(P, \mu), (P', \mu')\rangle), v', t')$ to $\mathsf{NAPS}^{\delta}_{\mathcal{G}}$. Once the transfer is finished, party P is informed about how many coins were transferred within this push request. This is done via a message $\mathsf{i\text{-}acknowledged}(\langle(P, \mu), (P', \mu')\rangle), v'')$, where v'' specifies the amount of coins that were transferred. If there are no open push requests of a form $(\pi || \langle(P, \mu), (P', \mu')\rangle, v, t)$ then a party P can decide to close a given push request by sending a message $\mathsf{i\text{-}acknowledge}((\pi || \langle(P, \mu), (P', \mu')\rangle, v, t)$ to $\mathsf{NAPS}^{\delta}_{\mathcal{G}}$. The function *remaining* and the accounts in the $P \multimap\!\!\circ\, P'$ channels are updated accordingly (by sending messages to the $\mathsf{Accounts}_{\mathcal{G}}$ functionality). If P_1 wants to finish processing given payment μ (this is possible only if there are no open push requests corresponding to μ other than the request $(\langle(P, \mu)\rangle, v, t)$) then she sends an $\mathsf{acknowledge}$ message to $\mathsf{NAPS}^{\delta}_{\mathcal{G}}$. The "ideal model" adversary will also be called the *simulator* and denoted $\mathcal{S}$. We assume that $\mathcal{S}$ has access to the ideal functionality. The *output of the ideal execution of* $\mathsf{NAPS}^{\delta}_{\mathcal{G}}$ against $\mathcal{S}$ and $\mathcal{Z}$ with security parameter 1^{κ} is a random variable $\mathsf{Ideal}(\mathsf{NAPS}^{\delta}_{\mathcal{G}}, \mathcal{S}, \mathcal{Z}, 1^{\kappa})$ denoting the output of $\mathcal{Z}$ In the extended version of this paper [7] we argue why the informal properties from Sec. 2.1.1 are implied by this ideal functionality.

The ideal functionality $\mathsf{NAPS}^{\delta}_{\mathcal{G}}$ is parametrized by a channel graph $\mathcal{G} = (\mathcal{P}, \mathcal{E}, \Gamma)$ and an arity parameter δ. It maintains a cash function $\widehat{\Gamma}$ initially equal to Γ and a set Ψ initially equal to $\emptyset$. Function $\widehat{\Gamma}$ is used to denote the current amount of coins available in the channels and set Ψ containing all *open push requests*. Moreover, the ideal functionality maintains a function $\mathit{remaining} : \Psi \to \mathbb{Z}_{\geq 0}$. It proceeds as follows.

Upon receiving a message of a form $(\mathsf{i\text{-}send}, \mu, u, t)$ from P_1 and $(\mathsf{i\text{-}receive}, \mu, u, t)$ from P_n (in the same round) – check if the following holds:

> *Correctness condition:* (a) you have not received an "i-send" or an "i-receive" message with this μ before and (b) the current time is greater than $t - \Delta$.

If it does not hold, then ignore this message. Otherwise (a) add $(\langle (P, \mu) \rangle, u, t)$ to Ψ and (b) let $\mathit{remaining}(\langle (P, \mu) \rangle, u, t) := v$.

Upon receiving a message of a form $(\mathsf{i\text{-}push}, (\pi || \langle (P, \mu), (P', \mu') \rangle), v, t)$ from P – check if the following holds:

> *Correctness condition:* (a) you have not received an "i-pushed" message with this $\langle (P, \mu), (P', \mu') \rangle$ before, (b) $P' \multimap\!\!\circ\, P \in \mathcal{E}$, (c) $v \leq \widehat{\Gamma}^{P \multimap\circ P'}(P)$, (d) the number of elements $(\pi || \langle (P, \mu), (P', \mu'), (P'', \mu'') \rangle), v', t')$ in Ψ (for any P'', μ'', v', and t) is less than δ, (e) the current time is greater than $t - \Delta$, and (f) if P is honest then $(\pi || \langle (P, \mu) \rangle) \in \Psi$ and $\mathit{remaining}(\pi || \langle (P, \mu) \rangle) \geq v$.

If it does not hold, then ignore this message. Otherwise: (a) add $(\pi || \langle (P, \mu), (P', \mu') \rangle), v, t)$ to Ψ, (b) decrement $\mathit{remaining}(\pi || \langle (P, \mu) \rangle), v, t)$ by v, (c) let $\mathit{remaining}(\pi || \langle (P, \mu), (P', \mu') \rangle), v, t) := v$, (d) decrement $\widehat{\Gamma}^{P \multimap\circ P'}(P)$ by v, and (e) in the next round send a message $(\mathsf{i\text{-}pushed}, (\pi || \langle (P, \mu), (P', \mu') \rangle), v, t))$ to P'.
If time $t + \Delta$ comes and $((\pi || \langle (P, \mu), (P', \mu') \rangle), v, t))$ is still in Ψ then behave as if you received a message $(\mathsf{i\text{-}acknowledge}, (\pi || \langle (P, \mu), (P', \mu) \rangle))$ from P' (see below).

Upon receiving a message of a form $(\mathsf{i\text{-}acknowledge}, (\pi || \langle (P, \mu) \rangle))$ from P – check if the following holds:

> *Correctness condition:* (a) $(\pi || \langle (P, \mu) \rangle, t, v) \in \Psi$ (for some v and t), and (b) there does not exist a push request $((\pi || \langle (P, \mu) \rangle || \langle (P', \mu') \rangle), v', t')$ in Ψ (for any P', μ', v', t').

If it does not hold, then ignore this message. Otherwise let v be the value from the "Correctness condition" and let s be the sum of the v' values in all the messages i -acknowledged$((\pi || \langle (P, \mu) \rangle || \langle (P', \mu') \rangle)), v')$ (for any (P', μ')) that were ever sent to P. If P_n is corrupt, allow the simulator to increase the value of s to any amount at most v.
Consider the following cases:

- $P = P_1$ (note that in this case π is empty) – then in the next round send $\mathsf{i\text{-}sent}(\langle (P, \mu) \rangle, s)$ to RVM.
- $P \in \{P_2, \ldots, P_{n-1}\}$ – then let (P_k, μ_k) be the last element of π and then within time Δ (a) send a message $(\mathsf{trans}, P_k, P, s)$ to $\mathsf{Accounts}_{\mathcal{G}}$, (b) increment $\widehat{\Gamma}^{P_k \multimap\circ P}(P_k)$ by $v - s$, (c) increment $\widehat{\Gamma}^{P_k \multimap\circ P}(P)$ by s, (d) increment $\mathit{remaining}(\pi, v, t)$ by $v - s$, (e) remove $(\pi || \langle (P, \mu) \rangle)$ from Ψ, and (f) send $\mathsf{i\text{-}acknowledged}((\pi || \langle (P, \mu) \rangle), s)$ to P_k.
- $P = P_n$ – then let (P_k, μ_k) be the last element of π and then within time Δ (a) send a message $(\mathsf{trans}, P_k, P, v)$ to $\mathsf{Accounts}_{\mathcal{G}}$, (b) increment $\widehat{\Gamma}^{P_k \multimap\circ P}(P_k)$ by v, (c) remove $(\pi || \langle (P, \mu) \rangle)$ from Ψ, and (d) send $\mathsf{i\text{-}acknowledged}((\pi || \langle (P, \mu) \rangle), s)$ to P_k.

Figure 2 The ideal functionality $\mathsf{NAPS}^{\delta}_{\mathcal{G}}$.

The real model. In the real model, the parties communicate with the environment and interact with each other directly. Before the protocol starts, we generate a (public key, secret key) pair for each P_i and give to P_i its secret key as input. Moreover, all parties (including RVM and $\mathcal{A}$) get the public keys of the other parties. For each pair $\{P_i, P_j\}$ such that $P_i \multimapboth P_j \in \mathcal{E}$ the parties P_i and P_j also have access to an uncorruptible *state channel machine* $C^{P_i \multimapboth P_j}$, which in turn, has access to $\mathsf{Accounts}_{\mathcal{G}}$ (the parties do not have a direct access to $\mathsf{Accounts}_{\mathcal{G}}$). Sending messages to $\mathsf{Accounts}_{\mathcal{G}}$ takes time at most Δ. The state channel machines and the parties know the public keys of all the parties. Altogether, a *NAPS protocol for a channel graph $\mathcal{G}$ with arity δ* is a tuple of machines $\Pi^{\delta}_{\mathcal{G}} := (RVM, P_1, \ldots, P_n, \{C^{P_i \multimapboth P_j}\}_{P_i \multimapboth P_j \in \mathcal{E}})$. The *output of the real execution of* Π with security parameter 1^{κ} is a random variable $\mathsf{Real}(\Pi^{\delta}_{\mathcal{G}}, \mathcal{A}, \mathcal{Z}, 1^{\kappa})$ denoting the output of $\mathcal{Z}$. We define security by requiring that no environment can distinguish between the ideal and the real model. In the definition, we use the concept of computational indistinguishably. From the construction of the ideal functionality, it is easy to see that all the informal security properties (fairness to the sender and the receiver and the balance neutrality) hold for EthNA.

▶ **Definition 1.** *A tuple* $\Pi^{\delta}_{\mathcal{G}}$ *is a* secure Non-Atomic Payment Splitting (NAPS) protocol for $\mathcal{G}$ and δ *if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that and every $\mathcal{Z}$ the families of random variables* $\{\mathsf{Ideal}(\mathsf{NAPS}^{\delta}_{\mathcal{G}}, \mathcal{S}, \mathcal{Z}, 1^{\kappa})\}_{\kappa}$ *and* $\{\mathsf{Real}(\Pi^{\delta}_{\mathcal{G}}, \mathcal{A}, \mathcal{Z}, 1^{\kappa})\}_{\kappa}$ *are computationally indistinguishable*

3.2 Formal description of EthNA

Let us start by providing formal definitions of some of the terms already informally introduced in Sec. 2.2. For a graph $\mathcal{G}$ and a nonce μ, a *subreceipt (over $\mathcal{G}$, for payment μ)* is a pair $\wr\pi, \lambda\int_{P_n}$ signed by P_n such that π is a path over $\mathcal{G}$ (for payment μ) with P_n appearing on the last position of π, and λ is a non-increasing sequence of positive integers, such that $|\lambda| = |\pi|$. A *payment report for μ* is a set $\mathcal{W}$ of subreceipts for μ such that π identifies a member of $\mathcal{W}$ uniquely, i.e.: ($\wr\pi, \lambda\int_{P_n} \in \mathcal{W}$ and $\wr\pi, \lambda'\int_{P_n} \in \mathcal{W}$) implies $\lambda = \lambda'$. For example, α, β, and γ in Sec. 2.2 are subreceipts, and the set $\{\beta, \gamma\}$ (see Eq. (4)) is a payment report (except that in that informal description, we omitted the nonces). For a payment report $\mathcal{W}$ a subreceipt $\wr(\pi, \lambda\int_{P_n}$ is a *leader of $\mathcal{W}$ at node P* if P appears on π at some position i, and for every $\wr\pi', \lambda'\int_{P_n} \in \mathcal{W}$ we have that $\lambda[i] \geq \lambda'[i]$. This notion was already discussed in Sec. 2.2, where in particular, we said that the leader of a payment report $\{\alpha', \gamma\}$ (on Eq. (4)) is γ. In normal cases (i.e. if P_n is honest), the leader is always unique and is equal to the *last* subreceipt of a from $\wr(\pi||\sigma'), \lambda'\int_{P_n}$ signed by P_n, however in general this does not need to be the case. When we talk about *the* leader of $\mathcal{W}$ at P we mean the leader that is the smallest according to some fixed linear ordering.

As mentioned in Sec. 1.2, EthNA is constructed using fraud-proofs. Formally, a *fraud-proof (for μ)* is a payment report $\mathcal{Q}$ for μ of a form $\mathcal{Q} = \{\wr(\sigma||\pi_i), \lambda_i\int\}_{i=1}^{m}$, where all the $\pi_i[1]$'s are pairwise distinct , such that the following condition holds: $\max_{i:=1,\ldots,m} \lambda_i[|\sigma|] < \sum_{i:=1}^{m} \lambda_i[|\sigma| + 1]$. For an example of a fraud-proof (with nonce missing from the picture) see Eq. (6). If EthNA has arity at most δ (see Sec. 2.1), then we require that $m \leq \delta$. Informally speaking, these conditions mean simply that in $\mathcal{Q}$ the largest label of σ is smaller than the sum of all labels of σ's children. If none of the subsets of a payment report $\mathcal{W}$ is a fraud-proof, then we say that $\mathcal{W}$ *is consistent.* As we show later, if P_n is honest, then $\mathcal{W}$ is always consistent. Note that the description of set $\mathcal{Q}$ as defined above can be quite large (it is of size $O(\delta \cdot (\ell + \kappa))$, where δ is EthNA's arity, ℓ is the maximal length of paths, and κ is the security parameter (we need this to account for the signature size). Luckily, there is a simple way to "compress" it to $O(\delta \cdot \kappa)$ (where κ is the security parameter) by exploiting the fact that the only values that are needed to prove cheating are the positions on the indices $|\sigma|$ and $|\sigma| + 1$ of the λ's. We describe the compression ideas in the extended version of this paper [7] .

The formal description of ETHNA appears on Figs. 3, 4, and 5. It uses a subroutine algorithm Add_Φ that we describe in a moment. We outsource some of the protocol to a procedure handle-path (depicted in the same figure) to avoid repeating the same instructions. The receipt verification machine RVM is presented in Fig. 6

Party P_1

Upon receiving an admissible message of a form $(\mathsf{i\text{-}send}, \mu, u, t)$ from the environment $\mathcal{Z}$ and in the next round a message $(\mathsf{invoice}, \wr \mu, u, t \wr_{P_n})$ from P_n – store this message, and execute the $\mathsf{handle\text{-}path}(P_1, \langle (P_1, \mu) \rangle, v, t)$ procedure presented on Fig. 4. Let (R, v) be the output of this procedure. Send $(\mathsf{acknowledged}, \mu, (\wr u, \mu, t \wr_{P_n}, R))$ to RVM.

Party P_i for $i = 2, \dots, n-1$

Upon receiving a message of a form $(\mathsf{push}, (\wr (\pi || \langle (P, \mu), (P', \mu') \rangle), v, t \wr_P)$ from some party P – ignore this message if at least one of the following happens: (a) $P' \neq P_i$ or (b) $t > \tau + \Delta$ (where τ is the current time). Otherwise run the *path handling procedure* $\mathsf{handle\text{-}path}(P_i, (\pi || \langle (P', \mu'), (P, \mu) \rangle), v, t)$ presented on Fig. 4. Let (R, v') be the output of this procedure and send $(\mathsf{acknowledge}, \wr (\pi || \langle (P', \mu'), (P, \mu) \rangle), v, t \wr_{P_j}, R)$ to $C^{P \circ\!\!-\!\!\circ P_i}$.

Party P_n

Wait to receive admissible messages of a form $(\mathsf{i\text{-}receive}, \mu, u, t)$ from the environment $\mathcal{Z}$. Handle each of them as follows.

Otherwise let β^μ be an integer variable initially equal to u and send a message $(\mathsf{invoice}, \wr \mu, u, t \wr_{P_n})$ to P_1. Let Φ^μ be a variable containing a payment report that is initially empty. Then wait (until time t comes) to receive messages of the following form:

Message $(\mathsf{push}, \wr (\pi || \langle (P, \mu), (P_n, \mu') \rangle), v, t' \wr_P)$ from some party P (with $t' \leq t$) – send a message $(\mathsf{pushed}, (\pi || \langle (P, \mu), (P_n, \mu') \rangle), v, t')$ to $\mathcal{Z}$.
If within time t you receive a message $(\mathsf{i\text{-}acknowledge}(\pi || \langle (P, \mu), (P_n, \mu') \rangle)$ from $\mathcal{Z}$ and $v > \beta^\mu$ then execute $\mathsf{Add}_{\Phi^\mu}((\pi || \langle (P, \mu), (P_n, \mu') \rangle), v)$. Let λ be the output of this procedure. Send a message $(\mathsf{acknowledge}, \wr (\pi || \langle (P, \mu), (P_n, \mu') \rangle), v, t' \wr_P, \wr (\pi || \langle (P, \mu), (P_n, \mu') \rangle), \lambda \wr_{P_n})$ to RVM.

Figure 3 The ETHNA protocol for the parties.

The parties receive the "ideal model" messages (starting with a prefix "i-") from $\mathcal{Z}$. By saying that a message (received from $\mathcal{Z}$) is *admissible*, we mean that it satisfies the "correctness conditions" from Fig. 2. The push requests are executed by direct communication between the parties, and the payment acknowledgment is done via the state channel machines. Let us comment on the types of messages that are sent within the protocol (see also the cheat sheet in [7]). The messages that are used are: "push" to push a subpayment (the corresponding message sent by the channel to the other party is "pushed"), "acknowledge" to acknowledge a subpayment (the corresponding message is "acknowledged"). The value R contains either a subreceipt (this is the most common case), or a fraud-proof, or a message "'empty" denoting the fact that no subpayments have been acknowledged by P_n.

Let $\mathcal{W}^{\pi}$ be a variable containing a set of subreceipts that initially is empty and let $\omega^{\pi} := \delta$. Send $(\mathsf{i\text{-}pushed}, \pi, v, t)$ to $\mathcal{Z}$ and wait for the following messages forms from $\mathcal{Z}$:

Message $(\mathsf{i\text{-}push}, (\pi || \langle (P, \mu), (P', \mu') \rangle, v', t')$ (for some $v' \leq \alpha^{\pi}$ and μ and μ' and P' such that $P \circ\!\!-\!\!\circ P' \in \mathcal{E}$) – handle each such a message as follows. If $\omega^{\pi} = 0$ then ignore this message. Otherwise let $\alpha^{\pi} := \alpha^{\pi} - v'$ and decrease ω^{π} by 1. Then send a message $(\mathsf{push}, \lbag (\pi || \langle (P', \mu') \rangle), v', t' \rbag_{P})$ to P' and wait until round t to receive a message od one of the following forms:

- $(\mathsf{acknowledged}, (\pi || \langle (P', \mu') \rangle), \mathsf{empty})$ from $C^{P_i \circ\!-\!\circ P'}$ – then let $\alpha^{\pi} := \alpha^{\pi} + v'$ and send a message $(\mathsf{i\text{-}acknowledged}, (\pi || \langle (P', \mu') \rangle), 0)$ to $\mathcal{Z}$,
- $(\mathsf{acknowledged}, (\pi || \langle (P', \mu') \rangle), \lbag \psi, \lambda \rbag_{P_n})$, where ψ is such that $(\pi || \langle (P', \mu') \rangle)$ is a prefix of ψ – then store $\lbag \psi, \lambda \rbag_{P_n}$ in $\mathcal{W}^{\pi}$ by letting $\mathcal{W}^{\pi} := \mathcal{W}^{\pi} \cup \{ \lbag \psi, \lambda \rbag_{P_n} \}$.

Let $\widehat{v} := \lambda[|\pi| + 1]$. Let $\alpha^{\pi} := \alpha^{\pi} + v' - \widehat{v}$ and send $(\mathsf{i\text{-}acknowledged}, (\pi || \langle (P', \mu') \rangle), \widehat{v})$ to $\mathcal{Z}$, or

- $(\mathsf{acknowledged}, (\pi || \langle (P', \mu') \rangle), (\mathsf{fraud\text{-}proof}, w))$ – then store $(\mathsf{fraud\text{-}proof}, w)$ and send a message $(\mathsf{i\text{-}acknowledged}, (\pi || \langle (P', \mu') \rangle), v')$ to $\mathcal{Z}$.

Message $(\mathsf{i\text{-}acknowledge}, \pi)$ (or time t comes) – if you are still waiting in the procedure of handling some "$\mathsf{i\text{-}push}$" message (see above), then ignore this message. Otherwise, do the following

- If you stored $(\mathsf{fraud\text{-}proof}, w)$ (for some (P', μ')) or if $\mathcal{W}^{\pi}$ is inconsistent and w is the fraud-proof – then output $((\mathsf{fraud\text{-}proof}, w), v)$.
- Otherwise: if $\mathcal{W}^{\pi}$ is empty then output empty.
- Otherwise let $\lbag \psi, \lambda \rbag_{P_n}$ be the leader of $\mathcal{W}^{\pi}$ at $\widetilde{P}$, where $\widetilde{P}$ is the last party on π. Output $(\lbag \psi, \lambda \rbag_{P_n}, \lambda(|\pi|))$.

Figure 4 Path handling procedure $\mathsf{handle\text{-}path}(P, \pi, v, t)$.

Recall that the values of registers $\Gamma^{P_i \circ\!-\!\circ P_j}(P_i)$ and $\Gamma^{P_i \circ\!-\!\circ P_j}(P_j)$ are pre-loaded before the execution started. Wait for messages from P_i and P_j.

Upon receiving a message of a form $(\mathsf{acknowledge}, \lbag \pi, v, t \rbag_{P_k}, \mathsf{empty})$ from a party P (such that $\{P_k, P\} = \{P_i, P_j\}$) – send $(\mathsf{acknowledged}, \pi, \mathsf{empty})$ to P_k.

Upon receiving a message of a form $(\mathsf{acknowledge}, \lbag \pi, v, t \rbag_{P_k}, \lbag \psi, \lambda \rbag_{P_n})$ from a party P where (a) current time is at most t, (b) π is a path with a suffix $\langle (P_k, \mu), (P, \mu') \rangle$ (for some μ and μ'), (c) π is a prefix of ψ, (d) $\lambda[|\pi|] \leq \Gamma^{P_i \circ\!-\!\circ P_j}(P_k)$, and (e) $\{P_k, P\} = \{P_i, P_j\}$ – then send a message of a form $(\mathsf{trans}, P_k, P, \lambda[|\pi|])$ to $\mathsf{Accounts}_{\mathcal{G}}$ and a message $(\mathsf{acknowledged}, \pi, \lbag \psi, \lambda \rbag_{P_n})$ to P_k.

Upon receiving a message of a form $(\mathsf{acknowledge}, \lbag \pi, v, t \rbag_{P_k}, (\mathsf{fraud\text{-}proof}, w))$ from a party P where (a) current time is at most t, (b) π is a path with a suffix $\langle (P_k, \mu), (P, \mu') \rangle$ (for some μ, μ' and P_k), (c) $v \leq \Gamma^{P_i \circ\!-\!\circ P_j}(P_k)$, and (d) w is a fraud-proof – then send a message $(\mathsf{trans}, P_k, P, v)$ to $\mathsf{Accounts}_{\mathcal{G}}$ and send a message $(\mathsf{acknowledged}, \pi, (\mathsf{fraud\text{-}proof}, w))$ to P_k.

Figure 5 The ETHNA state channel machine $C^{P_i \circ\!-\!\circ P_j}$.

Upon receiving a message $(\mathsf{acknowledged}, \mu, (\wr u, \mu', t \wr_{P_n}, R))$ from P_1 (such that $\mu = \mu'$ and you have not received an "acknowledged" message with this μ from P_1 before) – let

$$w := \begin{cases} u & \text{if } R = (\mathsf{fraud\text{-}proof}, w), \\ 0 & \text{if } R = \mathsf{empty} \\ \lambda[1] & \text{if } R = \wr \mathsf{acknowledge}, \psi, \lambda \wr_{P_n}, \end{cases}$$

where w is a fraud-proof. Send $(\mathsf{i\text{-}sent}, \mu, w)$ to $\mathcal{Z}$

Figure 6 Receipt Verification Machine *RVM*.

As described above, the main tasks of each party P_i (for $i = 2, \dots, n-1$) are: (a) receive push requests from some P, (b) forward corresponding push request in the direction of P_n, (c) receive information about how many coins were transferred, and (d) once you are done with handling all push requests: check if you received or you can find a fraud-proof – if yes, then forward this information back to P (via the state channel), and if not, then choose the leader of the set of receipts and forward it back to P (via the state channel). The procedure for P_1 is similar, except that P_1 is activated by a "send" message from $\mathcal{Z}$, and waits of the invoice from P_n. It then communicates with the receipt verification machine defined on Fig. 6 (see p. 19 in the appendix). Probably the most interesting part is the instructions for P_n. First, P_n (upon receiving an $\mathsf{i\text{-}receive}(\mu, u, t)$ message from $\mathcal{Z}$) sends an invoice to P_1. For every payment, μ party P_n maintains a payment tree Φ^{μ} that is initially empty. Payment trees were already discussed in Sec. 2.2. For a formal definition, consider some fixed μ and $\mathcal{G}$. During the execution of ETHNA for $\mathcal{G}$ and μ, several subpayments are delivered to P_n. Let $\pi^1, \dots, \pi^t$ denote the consecutive paths over which these subpayments go (of course, they need to be distinct), and let $v^i \in \mathbb{Z}_{>0}$ be the number of coins transmitted with each π^i. Let $\mathcal{W} := \{(\pi^i, v^i)\}_{i=1}^{t}$. Formally, a *payment tree* $\mathsf{tree}(\mathcal{W})$ is a labeled tree $(T, \mathcal{L})$, where T is the set of all prefixes of the π^i's, i.e., $T := \bigcup_i \mathsf{prefix}(\pi^i)$. . If ETHNA has arity δ then the arity of T in every node $(\pi || \langle (P, \mu) \rangle)$ is at most δ. Then for every $\pi \in T$ we let $\mathcal{L}(\pi) := \sum_{i : \pi \in \mathsf{prefix}(\pi^i)} v^i$. In other words: every path π gets labeled by the arithmetic sum of the value of the payments that were "passed through it". Clearly, the label $\mathcal{L}(\varepsilon)$ of the root node of $\mathsf{tree}(\mathcal{W})$ is equal to the sum of all v^i's, and hence it is equal to the total number of coins transferred by the subpayments in $\mathcal{W}$. We also have that for every path σ $\mathcal{L}(\sigma) = \sum_{\pi \text{ is a child of } \sigma} \mathcal{L}(\pi)$. It is also easy to see that $\mathsf{tree}(\mathcal{W})$ can be constructed "dynamically" by processing elements of $\mathcal{W}$ one after another. More precisely, this is done as follows. We start with an empty tree Φ, and then iteratively apply the algorithm Add_{Φ} (see Alg. 1) for $(\pi^1, v^1), (\pi^2, v^2), \dots$. From the construction of the algorithm, it follows

Algorithm 1 $\mathsf{Add}_{\Phi}(\pi, v)$.

```
This algorithm operates on a global state $\Phi = (T, \mathcal{L})$. Its side effect is a change of the global
state. We assume that $v \in \mathbb{Z}_{>0}$ and $\pi \notin T$.
for $j = 1, \dots, |\pi|$ do
    if $\pi|_j \in T$ then
        let $\mathcal{L}(\pi|_j) := \mathcal{L}(\pi|_j) + v$
    else
        let $T := T \cup \{\pi|_j\}$ let $\mathcal{L}(\pi|_j) := v$
output $\langle \mathcal{L}(\pi[1]), \dots, \mathcal{L}(\pi_{|\pi|}) \rangle$ (the labels on path $\pi$)
```

immediately that if P_n starts with Φ being an empty tree, and then iteratively applies Add_Φ to (π^i, v^i)'s for $i = 1, \ldots, t$, then the final state of Φ is equal to $\mathsf{tree}(\mathcal{W})$. For example, if P_n applies this procedure to the situation in Fig. 1 she obtains the trees depicted on Eqs. (1)–(3). It is easy to see that if P_n applies the Add_Φ algorithm correctly, then the resulting sets $\mathcal{W}$ are never inconsistent (and hence no fraud-proof will ever be produced against an honest P_n).

The formal security analysis of this protocol is given in the following lemma, whose proof appears in [7].

▶ **Lemma 2.** *Assuming that the underlying signature scheme is existentially unforgeable under a chosen-message attack,* EthNA *is a secure NAPS protocol for every $\mathcal{G}$ and δ.*

3.3 Efficiency analysis

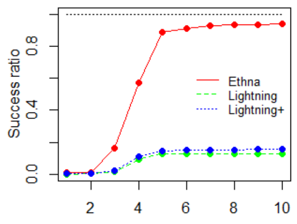
We consider separately the *optimistic* scenario (when the parties are cooperating) and the *pessimistic* one when the malicious parties slow down the execution. In the optimistic case, the payments are almost immediate. It takes 1 round for a payment to be pushed and 2 rounds to be acknowledged (due to the communication with the state channel machine). Hence, in the most optimistic case, the time for executing a payment is $3 \cdot \ell$ (where ℓ is the depth of the payment tree). During the acknowledgment, every malicious party can delay the process by time at most Δ. Hence, the maximal pessimistic time is $(1 + \Delta) \cdot \ell$. The second important measure is the blockchain costs, i.e., the fees that the parties need to pay. Below we provide a "theoretical" analysis of such costs. For the results of concrete experiments, see . Note that in the optimistic case, the only costs are channel opening and closing, and hence they are independent of the tree depth and of its arity. In the pessimistic case, all messages in state channels must be sent "via the blockchain". Let us consider two cases. In the first case, there is no fraud-proof. Then, the only message that is sent via the blockchain is $\mathsf{acknowledge}(\wr \phi, \lambda \int_{P_n})$, which has size linear $O(\ell + \kappa)$ (where ℓ is as above, and κ is the security parameter and corresponds to space needed to store a signature). The situation is a bit different if a fraud-proof appears. As remarked in Sec. 3.2 the size of a fraud-proof is $O(\delta \cdot (\ell + \kappa))$, where δ is EthNA's arity, ℓ is the maximal length of paths, and κ is the security parameter. Note that the fraud-proof is "propagated", i.e., even if a given intermediary decided to keep its arity small (i.e., not to split her subpayments into too many subpayments), she might be forced to pay fees that depend on some (potentially larger) arity. This could result in griefing attacks, which is why we introduced a global limit on the arity. There are many ways around this. First, we could modify the protocol so that the fraud-proofs by P_n are posted directly in a smart contract on a blockchain so that all other parties do not need to re-post and can just refer to it. Moreover, the proof size can be significantly reduced (see the extended version of this paper [7]).

4 Practical aspects

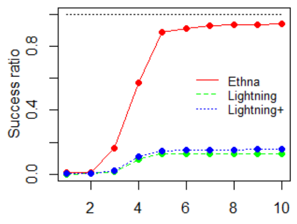
Let us now we provide information about practical experiments of EthNA implementation. We implemented a simple version of EthNA in Solidity. The source code is available at github.com/Sam16450/NAPS-EthNA. The table in Fig. 7 (a) summarizes the execution costs *in terms of thousands of gas*, and depending on the arity δ and the maximal path length. The `constructor` denotes the procedure for deploying a channel, `close` corresponds to closing a channel without disagreement, `addState` is used to register the balance in case of disagreement, `addCheatingProof` is used to add a fraud-proof, `addCompletedTransaction` – to add a subreceipt when no cheating was discovered, and `closeDisagreement` – to finally close a channel after disagreement.

δ	path length	constructor	close	addState	addCheatingProof	addCompletedTransaction	closeDisagreement
5	10	2,391	14	93	1,053	155	14
5	5	2,249	14	94	871	145	14
2	5	2,088	14	93	779	145	14
2	3	2,191	14	93	590	140	14

(a)



(b)



(c)

Figure 7 Experimental results.

Although routing algorithms are not the main topic of this work, we also performed some experiments with a routing algorithm built on top of ETHNA. We took the network graph in our experiments from the Lightning network (from the website gitlab.tu-berlin.de/rohrer/discharged-pc-data) with approx. 6K nodes and 30K channels. Channel's capacities are chosen according to the normal distribution $\mathcal{N}(200, 50)$. Each transaction was split by applying the following rules. The sender and the intermediaries look at the channel graph and search for the set $\mathcal{X}$ of shortest paths that lead to the receiver (and have different first elements). Then they split the payment into values proportional to the capacity of the first channel in the path. In our simulations, we performed 100K transaction. The results appear in Fig. 7. The "success ratio" denotes the probability of complete success of an average payment. Each transaction had to be completed in a maximum of 50 rounds. "Lightning" refers to standard Lightning routing, and "Lightning+" refers to the Lightning algorithm that attempts to push payments multiple times. Transaction values are chosen uniformly from set (x_0, x_1), while in (b) we have $(x_0, x_1) = (10, 500)$ and in (c) we have $x_0 := 150, 200, 300, 400$ and $x_1 := 500$. Our experiments show that even this simple routing algorithm for ETHNA works much better than Lightning.

5 Conclusions and future work

We have introduced a Non-Atomic Payment Splitting (NAPS) technique for the payment networks, constructed the ETHNA protocol that uses it, and proven its security. Due to the limited space, we focused only on introducing the payment-splitting technique. This paper opens several exciting questions for future research. First, it would be interesting to develop routing algorithms that use this feature. Secondly, we did not address privacy and anonymity in this setting, and it would be interesting to explore this topic. Our solution strongly relies on the Turing completeness of the underlying blockchain platform. It would be interesting to examine if NAPS schemes can be implemented Non-Atomic also over legacy blockchains such as Bitcoin.

References

1 Vivek Kumar Bagaria, Joachim Neu, and David Tse. Boomerang: Redundancy improves latency and throughput in payment-channel networks. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 304–324. Springer, 2020. doi:10.1007/978-3-030-51280-4_17.

2 Manuel M. T. Chakravarty, Sandro Coretti, Matthias Fitzi, Peter Gazi, Philipp Kant, Aggelos Kiayias, and Alexander Russell. Fast isomorphic state channels. In Nikita Borisov and Claudia Díaz, editors, *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II*, volume 12675 of *Lecture Notes in Computer Science*, pages 339–358. Springer, 2021. doi:10.1007/978-3-662-64331-0_18.

3 Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In Andrzej Pelc and Alexander A. Schwarzmann, editors, *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*, volume 9212 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-21741-3_1.

4 Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. Multi-party virtual state channels. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 625–656. Springer, 2019. doi:10.1007/978-3-030-17653-2_21.

5 Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 106–123. IEEE, 2019. doi:10.1109/SP.2019.00020.

6 Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 949–966. ACM, 2018. doi:10.1145/3243734.3243856.

7 Stefan Dziembowski and Pawel Kędzior. Non-atomic payment splitting in channel networks. *IACR Cryptol. ePrint Arch.*, 2020. URL: https://eprint.iacr.org/2020/166.

8 Lisa Eckey, Sebastian Faust, Kristina Hostáková, and Stefanie Roos. Splitting payments locally while routing interdimensionally. *IACR Cryptol. ePrint Arch.*, 2020. URL: https://eprint.iacr.org/2020/555.

9 Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 473–489. ACM, 2017. doi:10.1145/3133956.3134093.

10 Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 201–226. Springer, 2020. doi:10.1007/978-3-030-51280-4_12.

11 Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. A composable security treatment of the lightning network. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*, pages 334–349. IEEE, 2020. doi:10.1109/CSF49147.2020.00031.

12 Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 839–858. IEEE Computer Society, 2016. `doi:10.1109/SP.2016.55`.

13 Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 455–471. ACM, 2017. `doi:10.1145/3133956.3134096`.

14 Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. URL: `https://www.ndss-symposium.org/ndss-paper/anonymous-multi-hop-locks-for-blockchain-scalability-and-interoperability/`.

15 Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, pages 508–526. Springer, 2019. `doi:10.1007/978-3-030-32101-7_30`.

16 Olaoluwa Osuntokun. [lightning-dev] amp: Atomic multi-path payments over lightning. `https://tinyurl.com/29m2d7wr`, 2018.

17 Dmytro Piatkivskyi and Mariusz Nowostawski. Split payments in payment networks. In Joaquín García-Alfaro, Jordi Herrera-Joancomartí, Giovanni Livraga, and Ruben Rios, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings*, volume 11025 of *Lecture Notes in Computer Science*, pages 67–75. Springer, 2018. `doi:10.1007/978-3-030-00305-0_5`.

18 Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments, 2016.

19 Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. High throughput cryptocurrency routing in payment channel networks. In Ranjita Bhagwan and George Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 777–796. USENIX Association, 2020. URL: `https://www.usenix.org/conference/nsdi20/presentation/sivaraman`.

20 Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A^{2}l: Anonymous atomic locks for scalability in payment channel hubs. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1834–1851. IEEE, 2021. `doi:10.1109/SP40001.2021.00111`.

21 Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. *CoRR*, abs/1908.04756, 2019. `arXiv:1908.04756`.

Revisiting the Nova Proof System on a Cycle of Curves

Wilson D. Nguyen
Stanford University, CA, USA

Dan Boneh
Stanford University, CA, USA

Srinath Setty
Microsoft Research, Redmond, WA, USA

Abstract

Nova is an efficient recursive proof system built from an elegant folding scheme for (relaxed) R1CS statements. The original Nova paper (CRYPTO'22) presented Nova using a single elliptic curve group of order p. However, for improved efficiency, the implementation of Nova alters the scheme to use a 2-cycle of elliptic curves. This altered scheme is only described in the code and has not been proven secure. In this work, we point out a soundness vulnerability in the original implementation of the 2-cycle Nova system. To demonstrate this vulnerability, we construct a convincing Nova proof for the correct evaluation of 2^{75} rounds of the Minroot VDF in only 116 milliseconds. We then present a modification of the 2-cycle Nova system and formally prove its security. The modified system also happens to be more efficient than the original implementation. In particular, the modification eliminates an R1CS instance-witness pair from the recursive proof. The implementation of Nova has now been updated to use our optimized and secure system. In addition, we show that the folding mechanism at the core of Nova is malleable: given a proof for some statement z, an adversary can construct a proof for a related statement z', at the same depth as z, without knowledge of the witness for z'.

2012 ACM Subject Classification Security and privacy → Cryptanalysis and other attacks

Keywords and phrases Cryptographic Protocols, Recursive Proof Systems, Folding, Vulnerability

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.18

Related Version *Full Version*: `https://eprint.iacr.org/2023/969`

Supplementary Material *Software (Demo Code)*: `https://github.com/MercysJest/NovaBreakingTheCycleAttack`, archived at `swh:1:dir:ca6b1263b64aa9e429f687369b2df7a68bdc4a7f`

Funding This work was partially funded by NSF, DARPA, the Simons Foundation, and NTT Research. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

1 Introduction

In a recent work, Kothapalli, Setty, and Tzialla introduced an elegant folding scheme for relaxed R1CS statements [12]. The scheme leads to the Nova proof system: an efficient and succinct proof system for incrementally verifiable computation, or IVC [21]. This proof system has many applications in the blockchain space, such as verifiable delay functions [20], a Nova-based ZK virtual machine [15], and outsourced computation.

The description and analysis of Nova in [12] restricts itself to a single chain of incremental computation, namely a series of identical computation steps that produce an output which is fed directly into the next step. At every step, a single application of some function F is applied, and a statement about the validity of the prior step is folded into an ongoing statement of validity. We refer to this as a single *IVC chain*.

To improve efficiency, the implementation of Nova [14] uses a 2-cycle of elliptic curves. This leads to a proof system that uses two parallel IVC chains that must be linked together. Until this work, the 2-cycle Nova system was only described in the implementation code and there was no public proof of security.

In this paper, we present two security concerns that affect the 2-cycle Nova system. First, in Section 7 we describe a soundness issue that enables an attacker to produce proofs for false statements. For example, we compute a convincing proof for an evaluation of 2^{75} rounds of the Minroot VDF [10] in only 116 milliseconds on a single laptop. The core issue that this attack exploits is that the 2-cycle Nova system produces an IVC proof that contains an additional R1CS instance-witness pair that is not sufficiently constrained by the verifier.

To fix this issue we first formally describe the two IVC chains approach used in the Nova implementation. Instead of describing the original scheme, we present in Sections 4 and 5 a modified version of the system that fixes the vulnerability and results in a shorter IVC proof. We present the scheme as a compiler that compilers a Nova-like folding scheme into an IVC proof using a cycle of curves. In Section 6 we prove knowledge soundness of this modified system. We followed responsible disclosure best practice and coordinated patches with the Nova authors. The Nova implementation has now been updated [19] to use this optimized and secure system.

Second, in Section 8, we show that Nova's IVC proofs are malleable, which can lead to a security vulnerability in some applications. We also discuss strategies to mitigate this issue.

We begin by establishing in Sections 2 and 3 the terminology needed to describe the 2-cycle Nova system (i.e the 2-cycle Nova IVC Scheme).

2 Preliminaries

2.1 Incrementally Verifiable Computation (IVC)

Incrementally verifiable computation, or IVC, was introduced by Valiant [21]. For a function $\mathsf{F} : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}^a$, and some public values $z_0, z_i \in \{0,1\}^a$, an IVC scheme lets a prover generate a succinct proof that it knows auxiliary values $\mathsf{aux}_0, \ldots, \mathsf{aux}_{i-1} \in \{0,1\}^b$ such that

$$\begin{tikzcd}
& \mathsf{aux}_0 \arrow[d] & \mathsf{aux}_1 \arrow[d] & & \mathsf{aux}_{i-1} \arrow[d] & \\
z_0 \arrow[r] & \boxed{\mathsf{F}} \arrow[r] & \boxed{\mathsf{F}} \arrow[r] & \cdots \arrow[r] & \boxed{\mathsf{F}} \arrow[r] & z_i
\end{tikzcd}$$

The following definition gives the syntax and security properties for an IVC scheme. The prover $\mathcal{P}$ in this definition computes a proof for one step in the IVC chain. Iterating the prover will produce a proof π_i for the entire chain of length i.

▶ **Definition 1** (IVC [21]). *An* ***IVC Scheme*** *is a tuple of efficient algorithms* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *with the following interface:*

- $\mathsf{Setup}(1^\lambda, n) \to \mathsf{pp}$: *Given a security parameter* 1^λ*, a poly-size bound* $n \in \mathbb{N}$*, outputs public parameters* pp*.*
- $\mathcal{P}(\mathsf{pp}, \mathsf{F}, (i, z_0, z_i), \mathsf{aux}_i, \pi_i) \to \pi_{i+1}$: *Given public parameters* pp*, a function* $\mathsf{F} : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}^a$ *computable by a circuit of size at most* n*, an index* $i \in \mathbb{N}$*, an initial input* $z_0 \in \{0,1\}^a$*, a claimed output* $z_i \in \{0,1\}^a$*, advice* $\mathsf{aux}_i \in \{0,1\}^b$*, and an IVC proof* π_i*, outputs a new IVC proof* π_{i+1}*.*
- $\mathcal{V}(\mathsf{pp}, \mathsf{F}, (i, z_0, z_i), \pi_i) \to 0/1$: *Given public parameters* pp*, a function* F*, an index* i*, an initial input* z_0*, a claimed output* z_i*, and an IVC proof* π_i*, outputs* 0 *(reject) or* 1 *(accept).*

An IVC Scheme satisfies the following properties:

Completeness. *For every poly-size bound $n \in \mathbb{N}$, for every* pp *in the output space of* $\mathsf{Setup}(1^\lambda, n)$*, for every function* $\mathsf{F} : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}^a$ *computable by a circuit within the poly-size bound n, for every collection of elements* $(i \in \mathbb{N}, z_0, z_i \in \{0,1\}^a)$, $\mathsf{aux}_i \in \{0,1\}^b$*, and IVC proof* π_i*,*

$$\Pr\left[\begin{array}{c} \mathcal{V}(\mathsf{pp}, \mathsf{F}, (i, z_0, z_i), \pi_i) = 1 \\ \Downarrow \\ \mathcal{V}(\mathsf{pp}, \mathsf{F}, (i+1, z_0, z_{i+1}), \pi_{i+1}) = 1 \end{array} \quad : \quad \begin{array}{l} \pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{F}, (i, z_0, z_i), \mathsf{aux}_i, \pi_i), \\ z_{i+1} \leftarrow F(z_i, \mathsf{aux}_i) \end{array}\right] = 1$$

Knowledge Soundness. *Let $n \in \mathbb{N}$ be a poly-size bound and $\ell(\lambda)$ be a polynomial in the security parameter. Let $\mathcal{F}$ be an efficient function sampling adversary that outputs a function* $\mathsf{F} : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}^a$ *computable by a circuit within the poly-size bound n. Then for every efficient IVC prover P^*, there exists an efficient extractor $\mathcal{E}$ such that the probability*

$$\Pr\left[\begin{array}{c} \mathcal{V}(\mathsf{pp}, \mathsf{F}, (i, z_0, z_i), \pi_i) = 1 \\ \Big\Downarrow \\ z_i = \mathsf{F}(\mathbf{z}_{i-1}, \mathsf{aux}_{i-1}) \ \wedge \\ (i = 1 \ \Rightarrow \ \mathbf{z}_{i-1} = z_0) \ \wedge \\ (i > 1 \ \Rightarrow \ \mathcal{V}(\mathsf{pp}, \mathsf{F}, (i-1, z_0, \mathbf{z}_{i-1}), \pi_{i-1}) = 1) \end{array} \quad : \quad \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n), \\ \rho \leftarrow \{0,1\}^{\ell(\lambda)}, \\ \mathsf{F} \leftarrow \mathcal{F}(\mathsf{pp}; \rho), \\ (i, z_0, z_i, \pi_i) \leftarrow P^*(\mathsf{pp}; \rho), \\ (\mathbf{z}_{i-1}, \mathsf{aux}_{i-1}), \pi_{i-1} \leftarrow \mathcal{E}(\mathsf{pp}; \rho) \end{array}\right]$$

is greater than or equal to $1 - \mathsf{negl}(\lambda)$*.*

▶ Remark 2 (Full Extraction). Our definition of knowledge soundness implies other notions of IVC knowledge soundness, which require the extraction of all the auxiliary values in the execution chain [12, 11, 1, 2]. Informally, consider some ρ and pp sampled at random, and an adversary $P^*(\mathsf{pp}; \rho)$ that outputs a proof for i iterations of the IVC. Then the knowledge extractor $\mathcal{E}$ can be used to construct an IVC prover for a proof of $i-1$ iterations. Applying the definition again to the prover derived from $\mathcal{E}$ implies that there is a knowledge extractor $\mathcal{E}'$ that outputs a valid $(z_{i-2}, \mathsf{aux}_{i-2})$, π_{i-2} with all but negligible probability. We can repeat this argument inductively to extract a vector of auxiliary values $(\mathsf{aux}_{i-1}, \ldots, \mathsf{aux}_0)$ that shows that the z_i output by P^* is computed correctly from z_0. Note that if $\text{time}(\mathcal{E}) > c \cdot \text{time}(P^*)$ for some constant $c > 1$, then this argument only works for $O(\log \lambda)$ steps before the running time of the extractor becomes super-polynomial in λ. We use this sequential IVC model for consistency with the original Nova [12, 14]. In certain applications, a tree-like IVC proof system might be preferable.

▶ Remark 3 (Zero Knowledge). In some settings one also wants the IVC scheme to be zero knowledge, but in this writeup we focus on knowledge soundness of the scheme.

2.2 Committed Relaxed R1CS over a Ring

The Nova Proof system over a cycle of curves (2-cycle Nova system) makes use of two finite fields $\mathbb{F}_1$ and $\mathbb{F}_2$ simultaneously. As such, it is convenient to treat the primitives used in Nova as operating on the finite commutative ring $\mathrm{R} := \mathbb{F}_1 \times \mathbb{F}_2$, where addition and multiplication are defined component wise. That is, for $a = (a_1, a_2)$ and $b = (b_1, b_2)$ in R, we define $a + b = (a_1 + a_2, b_1 + b_2)$ and $a \cdot b = (a_1 b_1, a_2 b_2)$.

▶ **Definition 4** (Commitment Scheme). *Let* R *be a finite commutative ring. A commitment scheme for vectors over* R *is a pair of efficient algorithms* $(\mathsf{Setup}_{\mathsf{com}}, \mathsf{Commit})$ *with the following interface:*

- $\mathsf{Setup}_{\mathsf{com}}(1^\lambda, \mathrm{R}, n) \to \mathsf{pp}_{\mathsf{com}}$: *Given a security parameter* $1^\lambda \in 1^{\mathbb{N}}$*, a description of a ring* R*, and a poly-size bound* $n \in \mathbb{N}$*, outputs public parameters* $\mathsf{pp}_{\mathsf{com}}$*.*
- $\mathsf{Commit}(\mathsf{pp}_{\mathsf{com}}, x) \to c$: *Given public parameters* $\mathsf{pp}_{\mathsf{com}}$ *and input* $x \in \mathrm{R}^n$*, outputs a commitment* c*.*

These algorithms need to satisfy the following properties.

- *Binding: Let* $n \in \mathbb{N}$ *be a poly-size bound. For every efficient adversary* $\mathcal{A}$ *and for every finite commutative ring* R *whose size is at most exponential in* λ*,*

$$\Pr\left[\begin{array}{c} \mathsf{Commit}(\mathsf{pp}_{\mathsf{com}}, x_0) = \mathsf{Commit}(\mathsf{pp}_{\mathsf{com}}, x_1) \wedge \\ x_0 \neq x_1 \end{array} : \begin{array}{l} \mathsf{pp}_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^\lambda, \mathrm{R}, n) \\ (x_0, x_1) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{com}}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

- *Additively Homomorphic: Given two commitments* $c \leftarrow \mathsf{Commit}(\mathsf{pp}_{\mathsf{com}}, x)$*,* $c' \leftarrow \mathsf{Commit}(\mathsf{pp}_{\mathsf{com}}, x')$ *to vectors* $x,\ x' \in \mathrm{R}^n$ *(not necessarily distinct), there is an efficient homomorphism* $\oplus$ *on commitments such that* $c \oplus c' = \mathsf{Commit}(\mathsf{pp}_{\mathsf{com}}, x + x')$*.*
- *Succinct: For any* $x \in \mathrm{R}^n$*, the commitment* $c \leftarrow \mathsf{Commit}(\mathsf{pp}_{\mathsf{com}}, x)$ *must have size* $|c| \leq \mathsf{poly}(\lambda, \log(n))$*.*

▶ **Definition 5** (Committed Relaxed R1CS over a Ring). *Consider* $m, n, \ell \in \mathbb{N}$ *where* $m > \ell$ *and a finite commutative ring* R*. Further, consider a commitment scheme* Commit *for vectors over* R*, where* pp_W *and* pp_E *are commitment parameters for vectors of size* $m - \ell - 1$ *and* n *respectively.*

- *A* ***committed relaxed R1CS instance*** *is a tuple* $\mathbb{U} := (\bar{E}, s, \bar{W}, x)$*, where* $\bar{E}$ *and* $\bar{W}$ *are commitments,* $s \in \mathrm{R}$*, and* $x \in \mathrm{R}^\ell$*.*
- *A committed relaxed R1CS instance* $\mathbb{U} = (\bar{E}, s, \bar{W}, x)$ *is* ***satisfiable*** *with respect to an* ***R1CS constraint system*** $\mathsf{R1CS} := (A, B, C \in \mathrm{R}^{n \times m})$ *if there exist a relaxed witness* $\mathbb{W} := (E \in \mathrm{R}^n,\ W \in \mathrm{R}^{m-\ell-1})$ *such that*

$$\bar{E} = \mathsf{Commit}(\mathsf{pp}_E, E), \quad \bar{W} = \mathsf{Commit}(\mathsf{pp}_W, W), \quad \text{and} \quad (A \cdot Z) \circ (B \cdot Z) = s \cdot (C \cdot Z) + E$$

  where $Z = (W, x, s)$*. We refer to* E *as the* ***error vector*** *and* W *as the* ***extended witness****.*
- *An instance-witness pair* $(\mathbb{U}, \mathbb{W})$ ***satisfies*** *a constraint system* $\mathsf{R1CS}$ *if* $\mathbb{W}$ *is a satisfying relaxed witness for* $\mathbb{U}$*. An instance-witness pair* $(\mathbb{u}, \mathbb{w})$ *pair* ***strictly satisfies*** *an R1CS constraint system* $\mathsf{R1CS}$ *if* (1) *the pair satisfies* $\mathsf{R1CS}$ *and* (2) $\mathbb{u}.\bar{E} = \bar{0}$ *is the commitment to the zero vector and* $s = 1$*.*

▶ Remark 6 (Trivially Satisfiable Instance-Witness Pairs). A committed instance-witness pair $(\mathbb{U}_\perp, \mathbb{W}_\perp)$ will denote a trivially satisfying pair for an R1CS constraint system $\mathsf{R1CS}$ over R. In Nova [12], this pair is constructed by setting $E, W,$ and x to appropriately sized zero vectors, $\bar{E}, \bar{W}$ to be commitments to the zero vectors, and s equal to 0.

2.3 A Folding Scheme for Committed Relaxed R1CS over a Ring

Folding schemes give an efficient approach to IVC. In recent years, several works [2, 12, 11, 1, 13, 17] constructed efficient folding schemes for different problems. Nova [12] introduces an elegant folding scheme, for folding two committed relaxed R1CS instances and their witnesses. Nova's folding scheme is a public-coin, one-round interactive protocol that is made non-interactive in the random oracle model using the Fiat-Shamir transform. Additionally, Nova heuristically instantiates the random oracle with a concrete hash function and assumes that this heuristic produces a protocol that is knowledge sound. A similar assumption is used in other recursive proof systems [2, 11, 1].

▶ **Definition 7.** *A **Non-Interactive Folding Scheme for Committed Relaxed R1CS** consists of an underlying commitment scheme* $(\mathsf{Setup}_{\mathsf{com}}, \mathsf{Commit})$ *(Definition 4) for committed relaxed instances (Definition 5) and a tuple of efficient algorithms* $(\mathsf{Fold}_{\mathsf{Setup}}, \mathsf{Fold}_{\mathcal{K}}, \mathsf{Fold}_{\mathcal{P}}, \mathsf{Fold}_{\mathcal{V}})$ *with the following interface:*

- $\mathsf{Fold}_{\mathsf{Setup}}(1^\lambda, n) \rightarrow \mathsf{pp}$: *Given a security parameter* $1^\lambda \in 1^{\mathbb{N}}$*, a poly-size bound* $n \in \mathbb{N}$*, outputs public parameters* pp *which contain the description of a finite commutative ring* R *and commitment parameters* $\mathsf{pp}_{\mathsf{com}}$ *for vectors over* R *within the size bound* n*.*
- $\mathsf{Fold}_{\mathcal{K}}(\mathsf{pp}, \mathsf{R1CS}) \rightarrow (\mathsf{pk}, \mathsf{vk})$ *Given public parameters* pp*, an R1CS constraint system* $\mathsf{R1CS}$ *over* R *within the poly-size bound* n*, outputs proving key* pk *and verifier key* vk*.*
- $\mathsf{Fold}_{\mathcal{P}}\left(\mathsf{pk}, (\mathbb{u}, \mathbb{w}), (\mathbb{U}, \mathbb{W})\right) \rightarrow \left(\bar{\mathsf{T}}, (\mathbb{U}', \mathbb{W}')\right)$: *Given a proving key* pk*, two committed relaxed R1CS instance-witness pairs* $(\mathbb{u}, \mathbb{w}), (\mathbb{U}, \mathbb{W})$*, outputs a folding proof* $\bar{\mathsf{T}}$ *in the commitment space, and a new committed relaxed R1CS instance-witness pair* $(\mathbb{U}', \mathbb{W}')$*.*
- $\mathsf{Fold}_{\mathcal{V}}\left(\mathsf{vk}, \mathbb{u}, \mathbb{U}, \bar{\mathsf{T}}\right) \rightarrow \mathbb{U}'$: *Given a verification key* vk*, two committed relaxed R1CS instances* $\mathbb{u}, \mathbb{U}$*, and a folding proof* $\bar{\mathsf{T}}$*, outputs a new committed relaxed R1CS instance* $\mathbb{U}'$*.*

These algorithms need to satisfy the following properties:

Completeness. *For every poly-size bound* $n' \in \mathbb{N}$*, for every* pp *in the output space of* $\mathsf{Fold}_{\mathsf{Setup}}(1^\lambda, n')$*, for every poly-size* $m, n, \ell \in \mathbb{N}$ *where* $m > \ell$*,* $n' > m - \ell - 1$*,* $n' > n$*, for every R1CS constraint system* $\mathsf{R1CS} := (A, B, C \in \mathrm{R}^{n \times m})$*, for every committed relaxed instance-witness pair* $(\mathbb{u}, \mathbb{w}), (\mathbb{U}, \mathbb{W})$ *for* $\mathsf{R1CS}$*,*

$$\Pr\left[\begin{array}{c} \mathbb{U}' = \mathbb{U}'' \\ \wedge \\ (\mathbb{u}, \mathbb{w}), (\mathbb{U}, \mathbb{W}) \text{ satisfy } \mathsf{R1CS} \\ \implies (\mathbb{U}', \mathbb{W}') \text{ satisfies } \mathsf{R1CS} \end{array} : \begin{array}{l} (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Fold}_{\mathcal{K}}(\mathsf{pp}, \mathsf{R1CS}), \\ \left(\bar{\mathsf{T}}, (\mathbb{U}', \mathbb{W}')\right) \leftarrow \mathsf{Fold}_{\mathcal{P}}\left(\mathsf{pk}, (\mathbb{u}, \mathbb{w}), (\mathbb{U}, \mathbb{W})\right), \\ \mathbb{U}'' \leftarrow \mathsf{Fold}_{\mathcal{V}}\left(\mathsf{vk}, \mathbb{u}, \mathbb{U}, \bar{\mathsf{T}}\right) \end{array}\right] = 1$$

Knowledge Soundness. *Let* $n \in \mathbb{N}$ *be a poly-size bound and* $\ell(\lambda)$ *be a polynomial in the security parameter. For every efficient adversary* $\mathcal{P}^*$*, there exist an efficient extractor* $\mathcal{E}$ *such that the probability*

$$\Pr\left[\begin{array}{c} \mathbb{U}' = \mathsf{Fold}_{\mathcal{V}}(\mathsf{vk}, \mathbb{u}, \mathbb{U}, \bar{\mathsf{T}}) \wedge \\ (\mathbb{U}', \mathbb{W}') \text{ satisfies } \mathsf{R1CS} \\ \Downarrow \\ (\mathbb{u}, \mathbb{w}), (\mathbb{U}, \mathbb{W}) \text{ satisfy } \mathsf{R1CS} \end{array} : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Fold}_{\mathsf{Setup}}(1^\lambda, n), \\ \rho \leftarrow \{0,1\}^{\ell(\lambda)}, \\ \left(\mathsf{R1CS}, (\mathbb{u}, \mathbb{U}, \bar{\mathsf{T}}), (\mathbb{U}', \mathbb{W}')\right) \leftarrow \mathcal{P}^*(\mathsf{pp}; \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Fold}_{\mathcal{K}}(\mathsf{pp}, \mathsf{R1CS}), \\ (\mathbb{w}, \mathbb{W}) \leftarrow \mathcal{E}(\mathsf{pp}; \rho) \end{array}\right]$$

is $\geq 1 - \mathsf{negl}(\lambda)$*. In words, the definition of knowledge soundness states that if an adversary* $\mathcal{P}^*$ *can create a folded statement* $\mathbb{U}'$ *of two statements* $\mathbb{u}$ *and* $\mathbb{U}$ *and a satisfying witness* $\mathbb{W}'$ *for* $\mathbb{U}'$*, then an extractor* $\mathcal{E}$ *for* $\mathcal{P}^*$ *can produce satisfying witnesses* $\mathbb{w}$ *for* $\mathbb{u}$ *and* $\mathbb{W}$ *for* $\mathbb{U}$*.*

Collision resistance. The Nova construction also uses collision resistant hash functions. To be comprehensive, we define these next.

▶ **Definition 8** (Collision Resistant Hash Functions). *Let* R *be a finite commutative ring such that* $|\mathrm{R}| \approx 2^\lambda$. *A hash function for* R *is a pair of efficient algorithms* $(\mathsf{Setup_H}, \mathsf{H})$ *with the following interface*:

- $\mathsf{Setup_H}(1^\lambda, \mathrm{R}) \to \mathsf{pp_H}$: *Given a security parameter* $1^\lambda \in 1^{\mathbb{N}}$ *and a description of* R, *outputs public parameters* $\mathsf{pp_H}$.
- $\mathsf{H}(\mathsf{pp_H}, x) \to h$: *Given public parameters* $\mathsf{pp_H}$ *and input* $x \in \mathrm{R}^*$, *outputs a hash* $h \in \mathrm{R}$.

A hash function is ***collision resistant*** *if for every efficient adversary* $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} \mathsf{H}(\mathsf{pp_H}, m_0) = \mathsf{H}(\mathsf{pp_H}, m_1) \wedge \\ m_0 \neq m_1 \end{array} : \begin{array}{l} \mathsf{pp_H} \leftarrow \mathsf{Setup_H}(1^\lambda, \mathrm{R}) \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pp_H}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

3 The Nova Proof System over a Cycle of Curves: Preliminary Details

In this section and the next, we describe details about the underlying primitives in the 2-cycle Nova System [14]. Section 5 describes the explicit operation of the modified IVC verifier and modified IVC prover.

Cycle of Elliptic Curves. To reduce the number of constraints related to group operations, the implementation of Nova uses a cycle of elliptic curves for which the discrete log problem is hard. Specifically, the Nova implementation is generic over any cycle of elliptic curves that implements certain Rust traits (Nova implements those traits for the pasta cycle of two curves [16]). We denote the elliptic curve groups as $\mathbb{G}_1$ and $\mathbb{G}_2$. We refer to the *scalar field* of an elliptic curve group $\mathbb{G}$ as the field $\mathbb{F}$ whose order is $|\mathbb{G}|$, and the *base field* of $\mathbb{G}$ as the field $\mathbb{F}'$ over which the elliptic curve is defined (i.e. the points have the form $(x, y) \in \mathbb{F}' \times \mathbb{F}'$).

The group $\mathbb{G}_1$ has scalar field $\mathbb{F}_1$ and base field $\mathbb{F}_2$, while $\mathbb{G}_2$ has scalar field $\mathbb{F}_2$ and base field $\mathbb{F}_1$. Group operations for $\mathbb{G}_1$ can be efficiently expressed as constraints over the base field $\mathbb{F}_2$. Symmetrically, group operations for $\mathbb{G}_2$ can be efficiently expressed as constraints over the base field $\mathbb{F}_1$. The groups $\mathbb{G}_1$ and $\mathbb{G}_2$ will be the commitment spaces for Pedersen vector commitments for vectors over $\mathbb{F}_1$ and $\mathbb{F}_2$ respectively.

Groups and Rings. We define the ring $\mathrm{R} := \mathbb{F}_1 \times \mathbb{F}_2$ as the set of tuples with one element in $\mathbb{F}_1$ and another in $\mathbb{F}_2$. We can naturally define the ring operations as the component-wise field operations. Similarly, define the group $\mathbb{G} := \mathbb{G}_1 \times \mathbb{G}_2$ and it's group operation as the component-wise group operation.

Commitments. In Nova, the folding procedure requires additively homomorphic commitments to vectors over a field $\mathbb{F}$. Their specific construction [12] uses Pedersen vector commitments belonging to a group $\mathbb{G}$ of order $|\mathbb{F}|$, for which the discrete log problem is hard. Nova's implementation [14] is generic over the commitment scheme and one can supply a different commitment scheme for vectors, but we restrict our attention to Pedersen vector commitments in this paper.

We generalize the Pedersen vector commitment to the ring $\mathrm{R} := \mathbb{F}_1 \times \mathbb{F}_2$ by composing a Pedersen vector commitment over $\mathbb{F}_1$ with commitment space $\mathbb{G}_1$ and a Pedersen vector commitment over $\mathbb{F}_2$ with commitment space $\mathbb{G}_2$. We write $x^{(1)} \in \mathbb{F}_1^n$ and $x^{(2)} \in \mathbb{F}_2^n$ for the left and right projections of a vector $x \in \mathrm{R}^n$. Then, the commitment to x is a pair of commitments: a commitment to $x^{(1)} \in \mathbb{F}_1^n$ and a commitment to $x^{(2)} \in \mathbb{F}_2^n$. Concretely, this commitment to the vector $x \in \mathrm{R}^n$ will be an element in $\mathbb{G} := \mathbb{G}_1 \times \mathbb{G}_2$.

Committed relaxed instances. Consider two R1CS constraint systems

$$\mathsf{R1CS}^{(1)} := (A_1, B_1, C_1 \in \mathbb{F}_1^{n_1 \times m_1}) \quad \text{and} \quad \mathsf{R1CS}^{(2)} := (A_2, B_2, C_2 \in \mathbb{F}_2^{n_2 \times m_2})$$

defined over $\mathbb{F}_1$ and $\mathbb{F}_2$, respectively. A committed relaxed instance for $\mathsf{R1CS}^{(1)}$ is a tuple

$$\mathbb{U}^{(1)} := \left(\bar{\mathsf{E}}^{(1)},\ s^{(1)},\ \overline{\mathsf{W}}^{(1)},\ x^{(1)}\right) \qquad \text{where} \qquad \bar{\mathsf{E}}^{(1)}, \overline{\mathsf{W}}^{(1)} \in \mathbb{G}_1, \quad s^{(1)} \in \mathbb{F}_1, \quad x^{(1)} \in \mathbb{F}_1^{\ell_1}.$$

The corresponding relaxed witness $\mathbb{W}^{(1)} = (E^{(1)}, W^{(1)})$ has an error vector $E^{(1)} \in \mathbb{F}_1^{n_1}$ and extended witness $W^{(1)} \in \mathbb{F}_1^{m_1 - \ell_1 - 1}$. Symmetrically, a committed relaxed instance for $\mathsf{R1CS}^{(2)}$ is a tuple

$$\mathbb{U}^{(2)} := \left(\bar{\mathsf{E}}^{(2)},\ s^{(2)},\ \overline{\mathsf{W}}^{(2)},\ x^{(2)}\right) \qquad \text{where} \qquad \bar{\mathsf{E}}^{(2)}, \overline{\mathsf{W}}^{(2)} \in \mathbb{G}_2, \quad s^{(2)} \in \mathbb{F}_2, \quad x^{(2)} \in \mathbb{F}_2^{\ell_2}.$$

The corresponding relaxed witness $\mathbb{W}^{(2)} = (E^{(2)}, W^{(2)})$ has error vector $E^{(2)} \in \mathbb{F}_2^{n_2}$ and $W^{(2)} \in \mathbb{F}_2^{m_2 - \ell_2 - 1}$.

The two constraint systems $\mathsf{R1CS}^{(1)}$ over $\mathbb{F}_1$ and $\mathsf{R1CS}^{(2)}$ over $\mathbb{F}_2$ can be treated as a single constraint system $\mathsf{R1CS} := (A, B, C \in \mathrm{R}^{n \times m})$ over $\mathrm{R} := \mathbb{F}_1 \times \mathbb{F}_2$. The constraint systems $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ are simply the left and right projections of $\mathsf{R1CS}$. A strict projection of $\mathsf{R1CS}$ would require the dimensions of $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ to be identical to the dimensions of $\mathsf{R1CS}$. In practice, $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ can have different dimensions. When abstractly combining the constraint systems to obtain $\mathsf{R1CS}$, we can pad the systems with dummy rows and columns so that $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ have the same dimension. In particular, $m = \max(m_1, m_2)$, $n = \max(n_1, n_2)$, and $l = \max(l_1, l_2)$. Similarly, we can treat instance-witness pairs $(\mathbb{U}^{(1)}, \mathbb{W}^{(1)})$, $(\mathbb{U}^{(2)}, \mathbb{W}^{(2)})$ as the left and right projection of an instance-witness pair $(\mathbb{U}, \mathbb{W})$ for $\mathsf{R1CS}$.

Hash Functions. Hash functions $\mathsf{H}_1 : \mathbb{F}_1^* \to \mathbb{F}_1$ and $\mathsf{H}_2 : \mathbb{F}_2^* \to \mathbb{F}_2$ are collision resistant hash functions that take as input an arbitrary number of field elements and output a single field element which encodes the hash. In Nova, this single field element can be represented as a scalar whose bit representation is at most 250 bits long. Thus, the output hash has a unique representation in both fields, whose elements are 256 bits.[1]

Concretely, define $h_1 := \mathsf{H}_1(\dots)$ as the output of H_1 for some arbitrary input elements $(\dots) \in \mathbb{F}_1^*$. The hash can be expressed as $h_1 = \sum_{i \le 250} b_i^{(1)} \cdot (2^{(1)})^i$ where $2^{(1)} \in \mathbb{F}_1$ and for all $i \le 250$, $b_i^{(1)} \in \mathbb{F}_1$ are bits in $\{0, 1\}$. The hash output $h_1 := \sum_{i \le 250} b_i^{(1)} \cdot (2^{(1)})^i$ in $\mathbb{F}_1$ can be represented as an element h_1' in $\mathbb{F}_2$. To do so, define $h_1' := \sum_{i \le 250} b_i^{(2)} \cdot (2^{(2)})^i$ where for all i, the bit $b_i^{(2)} \in \mathbb{F}_2$ is the same the bit $b_i^{(1)} \in \mathbb{F}_1$ (i.e. if $b_i^{(1)} = 1^{(1)}$, we define $b_i^{(2)} = 1^{(2)}$ otherwise $b_i^{(2)} = 0^{(2)}$). Symmetrically, a hash output $h_2 := \sum_{i \le 250} b_i^{(2)} \cdot (2^{(2)})^i$ in $\mathbb{F}_2$ can be represented as an element h_2' in $\mathbb{F}_1$ in the same way.

Similarly, the hash function $\mathsf{H} : \{0, 1\}^* \to \{0, 1\}^\lambda$ is a collision resistant hash function whose outputs can be represented uniquely in both fields. We omit the hash parameters for H for ease of presentation. The Nova implementation [14] uses the Poseidon hash function [8] for H_1 and H_2 and SHA-3 [6] for H.

[1] The size of a digest is configurable, but a digest length of 250 bits was chosen to support a variety of popular curve cycles e.g., secp/secq, pallas/vesta (pasta curves), BN254/Grumpkin.

3.1 Folding over a Cycle of Curves

In Nova [12], a non-interactive folding scheme in the random oracle model is constructed by applying the Fiat-Shamir transform [7] to an interactive folding scheme. By instantiating the random oracle with an appropriate cryptographic hash function, they heuristically obtain a non-interactive folding scheme in the plain model. The construction described in [12] is limited to an R1CS constraint system $\mathsf{R1CS}$ defined over a field $\mathbb{F}$ with commitments belonging to a group $\mathbb{G}$ (with scalar field $\mathbb{F}$).

We extend the construction to R1CS constraint systems $\mathsf{R1CS}$ defined over a ring $\mathrm{R} := \mathbb{F}_1 \times \mathbb{F}_2$ by composing a folding scheme for R1CS constraint systems defined over $\mathbb{F}_1$ and a folding scheme for R1CS constraint systems defined over $\mathbb{F}_2$. When we fold committed relaxed instances for $\mathsf{R1CS}^{(1)}$, we implicitly mean run the folding scheme for systems over $\mathbb{F}_1$. Symmetrically, when we fold committed relaxed instances for $\mathsf{R1CS}^{(2)}$, we implicitly mean run the folding scheme for systems over $\mathbb{F}_2$. However, the random oracle calls used in both folding scheme will need to take in an argument vk, which is derived from both systems. We describe this in more detail in the description of $\mathsf{Fold}_{\mathcal{K}}$.

3.1.1 Folding Setup

$\mathsf{Fold}_{\mathsf{Setup}}$ takes in as input:

- A security parameter 1^λ.
- A poly-size bound $n \in \mathbb{N}$.

The algorithm performs the following steps:

1. Sample a cycle of elliptic curves $(\mathbb{G}_1, \mathbb{F}_1, \mathbb{G}_2, \mathbb{F}_2) \leftarrow \mathsf{SampleCycle}(1^\lambda)$.
2. Sample collision resistant hash parameters $\mathsf{pp}_{\mathsf{H}_1} \leftarrow \mathsf{Setup}_{\mathsf{H}}(1^\lambda, \mathbb{F}_1)$, $\mathsf{pp}_{\mathsf{H}_2} \leftarrow \mathsf{Setup}_{\mathsf{H}}(1^\lambda, \mathbb{F}_2)$.
3. Sample commit params $\mathsf{pp}_{\mathsf{com}_1} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^\lambda, \mathbb{F}_1, n)$ and $\mathsf{pp}_{\mathsf{com}_2} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^\lambda, \mathbb{F}_2, n)$. [2]
4. Output $\mathsf{pp} := \big((\mathbb{G}_1, \mathbb{F}_1, \mathbb{G}_2, \mathbb{F}_2),\ \ \mathsf{pp}_{\mathsf{H}_1}, \mathsf{pp}_{\mathsf{H}_2}, \mathsf{pp}_{\mathsf{H}},\ \ \mathsf{pp}_{\mathsf{com}_1}, \mathsf{pp}_{\mathsf{com}_2}\big)$.

3.1.2 Folding Keygen

$\mathsf{Fold}_{\mathcal{K}}$ takes in as input:

- Public parameters pp
- An R1CS constraint system $\mathsf{R1CS}$ over R within the poly-size bound n.

The algorithm performs the following steps:

1. Assign the verification key vk to a hash digest of the public parameters and constraint systems
$$\mathsf{vk} \leftarrow \mathsf{H}\big(\mathsf{pp},\ \mathsf{R1CS} := (\mathsf{R1CS}^{(1)}, \mathsf{R1CS}^{(2)})\big) \tag{1}$$
2. Assign the proving key $\mathsf{pk} \leftarrow \mathsf{pp}$ to be the public parameters.
3. Output $(\mathsf{pk}, \mathsf{vk})$.

The Verification Key. The Nova folding scheme is derived from an interactive protocol via the Fiat-Shamir transform [7]. As such, queries to the random oracle must include a description of the entire environment. Concretely, let H be an appropriate cryptographic hash function that heuristically instantiates a random oracle and whose outputs can be represented uniquely in both fields. The vk element (assigned in (1)) denotes a hash digest of the

[2] The commitment parameters pp_E, pp_W will be prefixes of $\mathsf{pp}_{\mathsf{com}}$ where the length is $\max\big(|E|,\ |W|\big)$.

environment. $\mathsf{Fold}_{\mathcal{V}}$ incorporates the elements $\mathsf{vk},\ \mathbb{u}, \mathbb{U}, \bar{\mathsf{T}}$ as arguments to its random oracle. We stress that this is needed to preserve the soundness of the Fiat-Shamir transform [4], as these digest elements represent inputs to the folding verifier when viewed as an interactive protocol.

4 The Augmented Constraint Systems Used in Nova

The 2-cycle Nova IVC Scheme operates on a pair of functions F_1 and F_2, one for each field. Abstractly, one can treat Nova as an IVC scheme for the combined function $\mathsf{F} : (\mathbb{F}_1^{a_1} \times \mathbb{F}_2^{a_2}) \times (\mathbb{F}_1^{b_1} \times \mathbb{F}_2^{b_2}) \to (\mathbb{F}_1^{a_1} \times \mathbb{F}_2^{a_2})$ of the form

$$\big((z^{(1)},\ z^{(2)}),\ (\mathsf{aux}^{(1)},\ \mathsf{aux}^{(2)})\big) \quad \overset{\mathsf{F}}{\longmapsto} \quad \big(\mathsf{F}_1(z^{(1)},\ \mathsf{aux}^{(1)}),\ \mathsf{F}_2(z^{(2)},\ \mathsf{aux}^{(2)})\big)$$

where $\mathsf{F}_1 : \mathbb{F}_1^{a_1} \times \mathbb{F}_1^{b_1} \to \mathbb{F}_1^{a_1}$ and $\mathsf{F}_2 : \mathbb{F}_2^{a_2} \times \mathbb{F}_2^{b_2} \to \mathbb{F}_2^{a_2}$ are poly-size arithmetic circuits over $\mathbb{F}_1$ and $\mathbb{F}_2$ respectively.

The 2-cycle Nova IVC scheme aims to prove that $(z_i^{(1)}, z_i^{(2)})$ is the result of iterating the function $\mathsf{F} = (\mathsf{F}_1, \mathsf{F}_2)$ a total of i times starting from the input $(z_0^{(1)}, z_0^{(2)})$ and using some auxiliary inputs. Every iteration of the IVC uses two R1CS constraint systems, one over $\mathbb{F}_1$ and one over $\mathbb{F}_2$, to verify that the functions F_1 and F_2 were evaluated correctly in that iteration. However, Nova augments these core constraint systems with additional constraints to verify that folding is done correctly at every iteration, and that the outputs of the previous iteration are properly forward to the current iteration. In this section we describe the two augmented constraint systems in detail.

The augmented constraint systems. The 2-cycle Nova IVC Scheme defines two augmented R1CS constraint systems $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ over $\mathbb{F}_1$ and $\mathbb{F}_2$. As noted in Section 3, a group operation for $\mathbb{G}_1$ can be efficiently expressed as constraints in the base field $\mathbb{F}_2$. Since the folding operation requires group operations in $\mathbb{G}_1$, the Nova implementation does the folding of the committed instances $\mathbb{u}^{(1)}$ and $\mathbb{U}^{(1)}$ for $\mathsf{R1CS}^{(1)}$ in the constraints of $\mathsf{R1CS}^{(2)}$. Symmetrically, the Nova implementation does the folding of the committed instances $\mathbb{u}^{(2)}$ and $\mathbb{U}^{(2)}$ for $\mathsf{R1CS}^{(2)}$ in the constraints of $\mathsf{R1CS}^{(1)}$.

The constraint systems $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ are defined as follows:

- let $\mathsf{R1CS}^{(1)}$ be the R1CS constraint system for the relation $\mathcal{R}_1$ defined in Figure 1a.
- let $\mathsf{R1CS}^{(2)}$ be the $\mathsf{R1CS}$ constraint system for the relation $\mathcal{R}_2$ defined in Figure 1b.

Intuitively, each constraint system applies one step of its function $z_{i+1} := \mathsf{F}(z_i,\ \mathsf{aux}_i)$, folds a prior committed instance $\mathbb{u}$ into a running committed instance $\mathbb{U}$ for the opposite constraint system, maintains the original inputs z_0, and updates the iteration index i. The public inputs $\mathbb{u}^{(1)}.x := (x_0, x_1)$ and $\mathbb{u}^{(2)}.x := (x_0, x_1)$ denote hashes that can be uniquely represented in both fields. We will explain these constraint systems in more detail when we describe the operation of the prover in Section 5.3.

Representation of Non-native Field elements and Arithmetic. Folding two committed instances $\mathbb{u}^{(1)}$ and $\mathbb{U}^{(1)}$ requires not only group operations over $\mathbb{G}_1$, but also field operations over $\mathbb{F}_1$. However, the R1CS constraint system $\mathsf{R1CS}^{(2)}$ over $\mathbb{F}_2$ has to encode the folding operation as constraints over $\mathbb{F}_2$. To account for this, $\mathbb{F}_1$ elements are encoded appropriately as $\mathbb{F}_2$ elements such that non-native arithmetic can be expressed as $\mathbb{F}_2$ constraints. The same strategy is symmetrically applied for folding constraints in $\mathsf{R1CS}^{(1)}$.

$$
\mathcal{R}_1 := \left\{ \begin{array}{l}
\left(\begin{array}{l}
\mathbb{u}^{(1)}_{i+1}.x := (x_0, x_1 \in \mathbb{F}_1)\ ; \\
\hat{w}^{(1)}_{i+1} := \big(\mathsf{vk} \in \mathbb{F}_1,\ \ i^{(1)} \in \mathbb{F}_1,\ \ z^{(1)}_0, z^{(1)}_i \in \mathbb{F}_1^{a_1}, \\
\qquad\qquad \mathsf{aux}^{(1)}_i \in \mathbb{F}_1^{b_1},\ \ \mathbb{U}^{(2)}_i, \mathbb{u}^{(2)}_i \in \mathcal{U}^{(2)},\ \ \bar{\mathsf{T}}^{(2)}_i \in \mathbb{G}_2\big) \\
\qquad \text{where } \mathcal{U}^{(2)} := \mathbb{G}_2 \times \mathbb{F}_2 \times \mathbb{G}_2 \times \mathbb{F}_2^2
\end{array} \right) : \\
\\
\text{If } i^{(1)} = 0^{(1)} : \\
\quad \text{Then set } \mathbb{U}^{(2)}_{i+1} := \mathbb{U}^{(2)}_{\perp} \\
\quad \text{Else set } \mathbb{U}^{(2)}_{i+1} := \mathsf{Fold}_{\mathcal{V}}\big(\mathsf{vk}, \mathbb{u}^{(2)}_i, \mathbb{U}^{(2)}_i, \bar{\mathsf{T}}^{(2)}_i\big) \\
\\
\text{Accept if} : \\
\quad \text{If } i^{(1)} = 0^{(1)} \text{ then } z^{(1)}_i = z^{(1)}_0 \\
\quad \mathbb{u}^{(2)}_i.\bar{E} = \bar{0}^{(2)} \\
\quad \mathbb{u}^{(2)}_i.s = 1^{(2)} \\
\quad \mathbb{u}^{(2)}_i.x_0 = \mathsf{H}_1\big(\mathsf{vk},\ i^{(1)},\ z^{(1)}_0,\ z^{(1)}_i,\ \mathbb{U}^{(2)}_i\big) \\
\quad x_0 = \mathbb{u}^{(2)}_i.x_1 \\
\quad x_1 = \mathsf{H}_1\big(\mathsf{vk},\ (i+1)^{(1)},\ z^{(1)}_0,\ z^{(1)}_{i+1} := \mathsf{F}_1(z^{(1)}_i,\ \mathsf{aux}^{(1)}_i),\ \mathbb{U}^{(2)}_{i+1}\big)
\end{array} \right\}
$$

(a) The relation $\mathcal{R}_1$ defining the R1CS constraint system $\mathsf{R1CS}^{(1)}$ on instance-witness pairs $\big(\mathbb{u}^{(1)}_{i+1}.x\ ;\ \hat{w}^{(1)}_{i+1}\big)$.

$$
\mathcal{R}_2 := \left\{ \begin{array}{l}
\left(\begin{array}{l}
\mathbb{u}^{(2)}_{i+1}.x := (x_0, x_1 \in \mathbb{F}_2)\ ; \\
\hat{w}^{(2)}_{i+1} := \big(\mathsf{vk} \in \mathbb{F}_2,\ \ i^{(2)} \in \mathbb{F}_2,\ \ z^{(2)}_0, z^{(2)}_i \in \mathbb{F}_2^{a_2}, \\
\qquad\qquad \mathsf{aux}^{(2)}_i \in \mathbb{F}_2^{b_2},\ \ \mathbb{U}^{(1)}_i, \mathbb{u}^{(1)}_{i+1} \in \mathcal{U}^{(1)},\ \ \bar{\mathsf{T}}^{(1)}_i \in \mathbb{G}_1\big) \\
\qquad \text{where } \mathcal{U}^{(1)} := \mathbb{G}_1 \times \mathbb{F}_1 \times \mathbb{G}_1 \times \mathbb{F}_1^2
\end{array} \right) : \\
\\
\text{If } i^{(2)} = 0^{(2)} : \\
\quad \text{Then set } \mathbb{U}^{(1)}_{i+1} := \mathbb{u}^{(1)}_{i+1} \\
\quad \text{Else set } \mathbb{U}^{(1)}_{i+1} := \mathsf{Fold}_{\mathcal{V}}\big(\mathsf{vk}, \mathbb{u}^{(1)}_{i+1}, \mathbb{U}^{(1)}_i, \bar{\mathsf{T}}^{(1)}_i\big) \\
\\
\text{Accept if} : \\
\quad \text{If } i^{(2)} = 0^{(2)} \text{ then } z^{(2)}_i = z^{(2)}_0 \\
\quad \mathbb{u}^{(1)}_{i+1}.\bar{E} = \bar{0}^{(1)} \\
\quad \mathbb{u}^{(1)}_{i+1}.s = 1^{(1)} \\
\quad \mathbb{u}^{(1)}_{i+1}.x_0 = \mathsf{H}_2\big(\mathsf{vk},\ i^{(2)},\ z^{(2)}_0,\ z^{(2)}_i,\ \mathbb{U}^{(1)}_i\big) \\
\quad x_0 = \mathbb{u}^{(1)}_{i+1}.x_1 \\
\quad x_1 = \mathsf{H}_2\big(\mathsf{vk},\ (i+1)^{(2)},\ z^{(2)}_0,\ z^{(2)}_{i+1} := \mathsf{F}_2(z^{(2)}_i,\ \mathsf{aux}^{(2)}_i),\ \mathbb{U}^{(1)}_{i+1}\big)
\end{array} \right\}
$$

(b) The relation $\mathcal{R}_2$ defining the R1CS constraint system $\mathsf{R1CS}^{(2)}$ on instance-witness pairs $\big(\mathbb{u}^{(2)}_{i+1}.x\ ;\ \hat{w}^{(2)}_{i+1}\big)$.

Hash parameters. The hash parameters $\mathsf{pp}_{\mathsf{H}_1}$ and $\mathsf{pp}_{\mathsf{H}_2}$ for H_1 and H_2 are hard-coded in the respective constraint systems. We omit the hash parameters in our paper for ease of notation, but implicitly call the hash function with their respective parameters generated in the IVC Setup.

Symmetry. If we omit the base case constraints, $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ are essentially symmetric constraint systems. The difference in indexing, $\mathbb{u}_i^{(2)}$ versus $\mathbb{u}_{i+1}^{(1)}$, is a notional choice that does not affect the symmetry. Additionally, we want to highlight that the only constraint on $\mathbb{u}_{i+1}^{(1)}.x_0$ and $\mathbb{u}_{i+1}^{(2)}.x_0$ are that they equal $\mathbb{u}_i^{(2)}.x_1$ and $\mathbb{u}_{i+1}^{(1)}.x_1$ respectively. As described in Section 3, hash values can be represented in both fields uniquely; thus, this equality is well-defined. Essentially, these *copy* constraints *pass* along the hashes meant for the public IO of the opposite instance. We will describe this strategy in more detail in Section 5.3.

5 The Modified Nova IVC Scheme

This section describes a modification to the prior (vulnerable) 2-cycle Nova proof system. In Section 6, we prove our modified system is knowledge sound (Definition 1).

5.1 Setup

The Nova Setup algorithm Setup takes in as input:
- A security parameter 1^λ.
- A poly-size bound $n \in \mathbb{N}$.

The algorithm outputs $\mathsf{pp} \leftarrow \mathsf{Fold}_{\mathsf{Setup}}(1^\lambda, n)$.

5.2 The Modified Nova Verifier

In this section, we describe a modified version of the 2-cycle Nova IVC verifier that patches a vulnerability found in the prior implementation. The algorithm is similar to the prior (vulnerable) 2-cycle Nova IVC verifier (Section 7.1), but the input IVC proof π_i omits a pair $(\mathbb{u}_i^{(1)}, \mathbb{w}_i^{(1)})$,which caused the original vulnerability. We provide a proof of knowledge soundness of our modified scheme in Section 6.

The Nova Verifier $\mathcal{V}$ takes in as input:
- IVC public parameters pp,
- a description of functions $\mathsf{F}_1 : \mathbb{F}_1^{a_1} \times \mathbb{F}_1^{b_1} \to \mathbb{F}_1^{a_1}$ and $\mathsf{F}_2 : \mathbb{F}_2^{a_2} \times \mathbb{F}_2^{b_2} \to \mathbb{F}_2^{a_2}$,
- an index $i \in \mathbb{N}$,
- starting values $z_0^{(1)} \in \mathbb{F}_1^{a_1}$ and $z_0^{(2)} \in \mathbb{F}_2^{a_2}$,
- claimed evaluations $z_i^{(1)} \in \mathbb{F}_1^{a_1}$ and $z_i^{(2)} \in \mathbb{F}_2^{a_2}$, and
- an IVC Proof for iteration i, namely $\pi_i := \big((\mathbb{u}_i^{(2)}, \mathbb{w}_i^{(2)}),\ (\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)}),\ (\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})\big)$.

The verifier first runs the following initial procedure, which can be treated as a preprocessing phase:
1. Given functions F_1 and F_2, deterministically generate augmented R1CS constraint systems $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ which implement relations $\mathcal{R}_1$ and $\mathcal{R}_2$ from Figures 1a and 1b
2. Compute the folding verification key

$$(\,\cdot\,, \mathsf{vk}) \leftarrow \mathsf{Fold}_{\mathcal{K}}\big(\mathsf{pp},\ \mathsf{R1CS} := (\mathsf{R1CS}^{(1)}, \mathsf{R1CS}^{(2)})\big)$$

Then, the verifier accepts if the following six conditions are met:

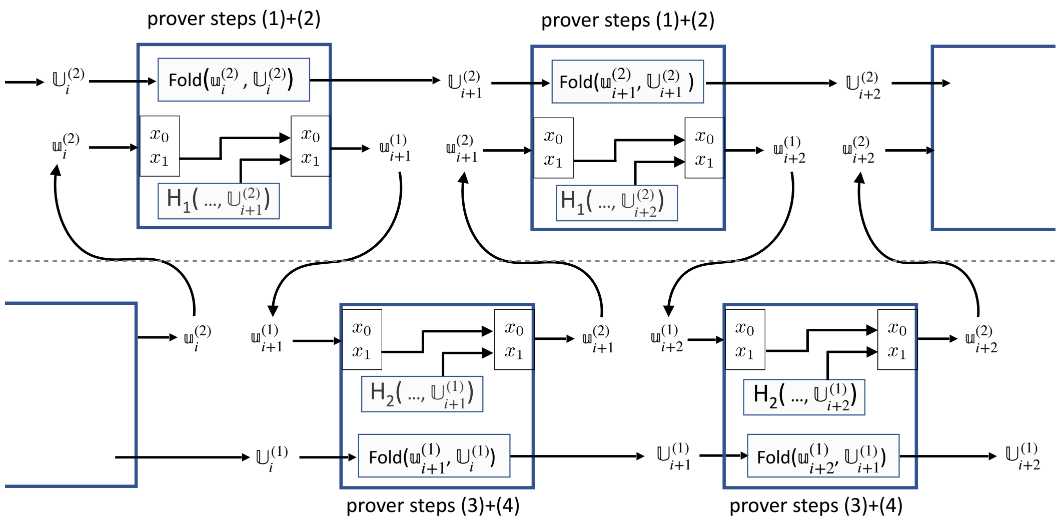


Figure 2 An illustration of the key parts of the prover's operation in the non-base case.

1. The index i must be greater than 0.
2. $\mathbb{u}_i^{(2)}.x_0 = \mathsf{H}_1\big(\mathsf{vk}, i^{(1)}, z_0^{(1)}, z_i^{(1)}, \mathbb{U}_i^{(2)}\big)$
3. $\mathbb{u}_i^{(2)}.x_1 = \mathsf{H}_2\big(\mathsf{vk}, i^{(2)}, z_0^{(2)}, z_i^{(2)}, \mathbb{U}_i^{(1)}\big)$
4. The pair $(\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)})$ satisfies $\mathsf{R1CS}^{(1)}$.
5. The pair $(\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})$ satisfies $\mathsf{R1CS}^{(2)}$.
6. The pair $(\mathbb{u}_i^{(2)}, \mathbb{w}_i^{(2)})$ **strictly** satisfies $\mathsf{R1CS}^{(2)}$.

5.3 The Modified Nova Prover

In this section, we describe a modified 2-cycle Nova IVC prover. The algorithm is similar to the prior 2-cycle Nova IVC prover, but the generated IVC proof π_{i+1} omits a pair $(\mathbb{u}_{i+1}^{(1)}, \mathbb{w}_{i+1}^{(1)})$, which caused the original vulnerability (Section 7.1). We first describe an initial procedure, then the base case step of the Nova prover, and then the recursive step as illustrated in Figure 2.

5.3.1 Initial Procedure

The prover performs an initial procedure identical to the initial procedure of the verifier:
1. Given functions F_1 and F_2, deterministically generate augmented R1CS constraint systems $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ which implement relations $\mathcal{R}_1$ and $\mathcal{R}_2$.
2. Compute the folding prover and verifier key

$$(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Fold}_{\mathcal{K}}\big(\mathsf{pp},\ \mathsf{R1CS} := (\mathsf{R1CS}^{(1)}, \mathsf{R1CS}^{(2)})\big)$$

5.3.2 The Base Case

The Nova Prover $\mathcal{P}$ takes in as input:

- IVC public parameters pp.
- A description of functions $\mathsf{F}_1 : \mathbb{F}_1^{a_1} \times \mathbb{F}_1^{b_1} \to \mathbb{F}_1^{a_1}$ and $\mathsf{F}_2 : \mathbb{F}_2^{a_2} \times \mathbb{F}_2^{b_2} \to \mathbb{F}_2^{a_2}$.
- Starting values $z_0^{(1)} \in \mathbb{F}_1^{a_1}$ and $z_0^{(2)} \in \mathbb{F}_2^{a_2}$.
- Auxiliary inputs $\mathsf{aux}_0^{(1)} \in \mathbb{F}_1^{b_1}$ and $\mathsf{aux}_0^{(2)} \in \mathbb{F}_2^{b_2}$.

The prover proceeds as follows:

- **Compute New Pair for $\mathsf{R1CS}^{(1)}$:** Compute the new committed pair $(\mathbb{u}_1^{(1)}, \mathbb{w}_1^{(1)})$ for $\mathsf{R1CS}^{(1)}$ as follows:
  - Define an initial dummy instance as $\mathbb{u}_0^{(2)} := \big(\bar{0}^{(2)},\ 1^{(2)},\ \bar{0}^{(2)},\ x := (x_0,\ x_1)\big)$ where
  $$x_0 := \mathsf{H}_1\big(\mathsf{vk},\ 0^{(1)},\ z_0^{(1)},\ z_0^{(1)},\ \mathbb{U}_\perp^{(2)}\big) \qquad \text{and} \qquad x_1 := \mathsf{H}_2\big(\mathsf{vk},\ 0^{(2)},\ z_0^{(2)},\ z_0^{(2)},\ \mathbb{U}_\perp^{(1)}\big)$$
  This instance will not be folded into any running instance.
  - Define $\hat{w}_1^{(1)} := (\mathsf{vk}, 0^{(1)}, z_0^{(1)}, z_0^{(1)}, \mathsf{aux}_0^{(1)}, \mathbb{U}_\perp^{(2)}, \mathbb{u}_0^{(2)}, \bar{0}^{(2)})$ as the relation witness for $\mathcal{R}_1$. Then, compute the extended witness $w_1^{(1)}$ by performing the computation on $\hat{w}_1^{(1)}$ required to satisfy the constraints expressed in $\mathsf{R1CS}^{(1)}$.
  - Commit to the extended witness $\bar{\mathsf{w}}_1^{(1)} \leftarrow \mathsf{Commit}\big(\mathrm{pp}_W^{(1)}, w_1^{(1)}\big)$.
  - Define $\mathbb{U}_1^{(2)} := \mathbb{U}_\perp^{(2)}$ and $\mathbb{W}_1^{(2)} := \mathbb{W}_\perp^{(2)}$.
  - Define $x_0 := \mathbb{u}_0^{(2)}.x_1$ and $x_1 := \mathsf{H}_1\big(\mathsf{vk}, 1^{(1)}, z_0^{(1)}, z_1^{(1)} := \mathsf{F}_1(z_0^{(1)}, \mathsf{aux}_0^{(1)}), \mathbb{U}_1^{(2)}\big)$.
  - Assign $\mathbb{u}_1^{(1)} := \big(\bar{0}^{(1)}, 1^{(1)}, \bar{\mathsf{w}}_1^{(1)}, (x_0, x_1)\big)$ and $\mathbb{w}_1^{(1)} := \big(\vec{0}^{(1)}, w_1^{(1)}\big)$.
- **Compute New Pair for $\mathsf{R1CS}^{(2)}$:** Compute the new committed pair $(\mathbb{u}_1^{(2)}, \mathbb{w}_1^{(2)})$ for $\mathsf{R1CS}^{(2)}$ as follows:
  - Define $\hat{w}_1^{(2)} := (\mathsf{vk}, 0^{(2)}, z_0^{(2)}, z_0^{(2)}, \mathsf{aux}_0^{(2)}, \mathbb{U}_\perp^{(1)}, \mathbb{u}_1^{(1)}, \bar{0}^{(1)})$ as the relation witness for $\mathcal{R}_2$. Then, compute the extended witness $w_1^{(2)}$ by performing the computation on $\hat{w}_1^{(2)}$ required to satisfy the constraints expressed in $\mathsf{R1CS}^{(2)}$.
  - Commit to the extended witness $\bar{\mathsf{w}}_1^{(2)} \leftarrow \mathsf{Commit}(\mathrm{pp}_W^{(2)}, w_1^{(2)})$.
  - Define $\mathbb{U}_1^{(1)} := \mathbb{u}_1^{(1)}$ and $\mathbb{W}_1^{(1)} := \mathbb{w}_1^{(1)}$.
  - Define $x_0 := \mathbb{u}_1^{(1)}.x_1$ and compute $x_1 := \mathsf{H}_2\big(\mathsf{vk}, 1^{(2)}, z_0^{(2)}, z_1^{(2)} := \mathsf{F}_2(z_0^{(2)}, \mathsf{aux}_0^{(2)}), \mathbb{U}_1^{(1)}\big)$.
  - Assign $\mathbb{u}_1^{(2)} := \big(\bar{0}^{(2)}, 1^{(2)}, \bar{\mathsf{w}}_1^{(2)}, (x_0, x_1)\big)$ and $\mathbb{w}_1^{(2)} := \big(\vec{0}^{(2)}, w_1^{(2)}\big)$.
- **Output Prover State:** Output IVC Proof for step 1
  $$\pi_1 := \big((\mathbb{u}_1^{(2)}, \mathbb{w}_1^{(2)}),\ (\mathbb{U}_1^{(1)}, \mathbb{W}_1^{(1)}),\ (\mathbb{U}_1^{(2)}, \mathbb{W}_1^{(2)})\big)$$
  along with new evaluations $z_1^{(1)} := \mathsf{F}_1(z_0^{(1)}, \mathsf{aux}_0^{(1)})$ and $z_1^{(2)} := \mathsf{F}_2(z_0^{(2)}, \mathsf{aux}_0^{(2)})$. These outputs are sufficient to execute another step of the Nova prover for iteration 1.

5.3.3 The Non-Base Case

The Nova Prover $\mathcal{P}$ takes in as input:

- IVC public parameters pp.
- Constraint Systems $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$.
- An index $i \in \mathbb{N}$, where $i \geq 1$.
- Starting values $z_0^{(1)} \in \mathbb{F}_1^{a_1}$ and $z_0^{(2)} \in \mathbb{F}_2^{a_2}$.
- Evaluations $z_i^{(1)} \in \mathbb{F}_1^{a_1}$ and $z_i^{(2)} \in \mathbb{F}_2^{a_2}$.
- Auxiliary inputs $\mathsf{aux}_i^{(1)} \in \mathbb{F}_1^{b_1}$ and $\mathsf{aux}_i^{(2)} \in \mathbb{F}_2^{b_2}$.
- An IVC Proof for Iteration i $\quad \pi_i := \big((\mathbb{u}_i^{(2)}, \mathbb{w}_i^{(2)}),\ (\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)}),\ (\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})\big)$.

The prover proceeds as follows (see also Figure 2):

1. **Fold Prior Pairs for $\mathsf{R1CS}^{(2)}$:** Fold the committed pairs $(\mathbb{u}_i^{(2)}, \mathbb{w}_i^{(2)})$ and $(\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})$ for $\mathsf{R1CS}^{(2)}$.

$$\mathsf{Fold}_{\mathcal{P}}\big(\mathsf{pk}, (\mathbb{u}_i^{(2)}, \mathbb{w}_i^{(2)}), (\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})\big) \rightarrow \big(\bar{\mathsf{T}}_i^{(2)}, (\mathbb{U}_{i+1}^{(2)}, \mathbb{W}_{i+1}^{(2)})\big)$$

Obtain a folding proof $\bar{\mathsf{T}}_i^{(2)}$ and new committed relaxed instance-witness pair $(\mathbb{U}_{i+1}^{(2)}, \mathbb{W}_{i+1}^{(2)})$.

2. **Compute New Pair for $\mathsf{R1CS}^{(1)}$:** Compute the new committed pair $(\mathbb{u}_{i+1}^{(1)}, \mathbb{w}_{i+1}^{(1)})$ for $\mathsf{R1CS}^{(1)}$ as follows:
   - Define $\hat{w}_{i+1}^{(1)} := (\mathsf{vk}, i^{(1)}, z_0^{(1)}, z_i^{(1)}, \mathsf{aux}_i^{(1)}, \mathbb{U}_i^{(2)}, \mathbb{u}_i^{(2)}, \bar{\mathsf{T}}_i^{(2)})$ as the relation witness for $\mathcal{R}_1$. Then, compute the extended witness $w_{i+1}^{(1)}$ by performing the computation on $\hat{w}_{i+1}^{(1)}$ required to satisfy the constraints expressed in $\mathsf{R1CS}^{(1)}$.
   - Commit to the extended witness $\bar{\mathsf{w}}_{i+1}^{(1)} \leftarrow \mathsf{Commit}(\mathrm{pp}_W^{(1)}, w_{i+1}^{(1)})$.
   - Define $x_0 := \mathbb{u}_i^{(2)}.x_1$ and $x_1 := \mathsf{H}_1\big(\mathsf{vk}, (i+1)^{(1)}, z_0^{(1)}, z_{i+1}^{(1)} := \mathsf{F}_1(z_i^{(1)}, \mathsf{aux}_i^{(1)}), \mathbb{U}_{i+1}^{(2)}\big)$.
   - Assign $\mathbb{u}_{i+1}^{(1)} := \big(\bar{0}^{(1)}, 1^{(1)}, \bar{\mathsf{w}}_{i+1}^{(1)}, (x_0, x_1)\big)$ and $\mathbb{w}_{i+1}^{(1)} := \big(\vec{0}^{(1)}, w_{i+1}^{(1)}\big)$.
3. **Fold Pairs for $\mathsf{R1CS}^{(1)}$:** Fold the newly computed pair $(\mathbb{u}_{i+1}^{(1)}, \mathbb{w}_{i+1}^{(1)})$ with the committed pair $(\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)})$ for $\mathsf{R1CS}^{(1)}$.

$$\mathsf{Fold}_{\mathcal{P}}\big(\mathsf{pk}, (\mathbb{u}_{i+1}^{(1)}, \mathbb{w}_{i+1}^{(1)}), (\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)})\big) \rightarrow \big(\bar{\mathsf{T}}_i^{(1)}, (\mathbb{U}_{i+1}^{(1)}, \mathbb{W}_{i+1}^{(1)})\big)$$

Obtain a folding proof $\bar{\mathsf{T}}_i^{(1)}$ and new committed relaxed instance-witness pair $(\mathbb{U}_{i+1}^{(1)}, \mathbb{W}_{i+1}^{(1)})$.

4. **Compute New Pair for $\mathsf{R1CS}^{(2)}$:** Compute the new committed pair $(\mathbb{u}_{i+1}^{(2)}, \mathbb{w}_{i+1}^{(2)})$ for $\mathsf{R1CS}^{(2)}$ as follows:
   - Define $\hat{w}_{i+1}^{(2)} := (\mathsf{vk}, i^{(2)}, z_0^{(2)}, z_i^{(2)}, \mathsf{aux}_i^{(2)}, \mathbb{U}_i^{(1)}, \mathbb{u}_{i+1}^{(1)}, \bar{\mathsf{T}}_i^{(1)})$ as the relation witness for $\mathcal{R}_2$. Then, compute the extended witness $w_{i+1}^{(2)}$ by performing the computation on $\hat{w}_{i+1}^{(2)}$ required to satisfy the constraints expressed in $\mathsf{R1CS}^{(2)}$.
   - Commit to the extended witness $\bar{\mathsf{w}}_{i+1}^{(2)} \leftarrow \mathsf{Commit}(\mathrm{pp}_W^{(2)}, w_{i+1}^{(2)})$.
   - Define $x_0 := \mathbb{u}_{i+1}^{(1)}.x_1$ and $x_1 := \mathsf{H}_2\big(\mathsf{vk}, (i+1)^{(2)}, z_0^{(2)}, z_{i+1}^{(2)} := \mathsf{F}_2(z_i^{(2)}, \mathsf{aux}_i^{(2)}), \mathbb{U}_{i+1}^{(1)}\big)$.
   - Assign $\mathbb{u}_{i+1}^{(2)} := \big(\bar{0}^{(2)}, 1^{(2)}, \bar{\mathsf{w}}_{i+1}^{(2)}, (x_0, x_1)\big)$ and $\mathbb{w}_{i+1}^{(2)} := \big(\vec{0}^{(2)}, w_{i+1}^{(2)}\big)$.
5. **Output Prover State:** Output IVC Proof for step $i+1$

$$\pi_{i+1} := \big((\mathbb{u}_{i+1}^{(2)}, \mathbb{w}_{i+1}^{(2)}),\ (\mathbb{U}_{i+1}^{(1)}, \mathbb{W}_{i+1}^{(1)}),\ (\mathbb{U}_{i+1}^{(2)}, \mathbb{W}_{i+1}^{(2)})\big)$$

along with new evaluations $z_{i+1}^{(1)} := \mathsf{F}_1(z_i^{(1)}, \mathsf{aux}_i^{(1)})$ and $z_{i+1}^{(2)} := \mathsf{F}_2(z_i^{(2)}, \mathsf{aux}_i^{(2)})$. These outputs are sufficient to execute another step of the Nova prover for iteration $i+1$.

This completes our description of the prover.

6 Proof of security

▶ **Theorem 9.** *If the non-interactive folding scheme is knowledge sound (Definition 7) and the hash function is collision resistant (Definition 8), then our modified Nova IVC scheme is knowledge sound (Definition 1).*

The proof of Theorem 9 can be found in the full version of our paper (eprint.iacr.org/2023/969).

7 The Original Nova Vulnerability

In this section, we describe the prior implementation of the Nova Verifier and the vulnerability in detail. At the end, we provide a proof of concept attack against the Minroot VDF [10] Nova verifier.

7.1 The Prior (Vulnerable) Nova Verifier

Before our patch added on 05/18/2023, the prior (vulnerable) 2-cycle Nova IVC Verifier $\mathcal{V}$ took in as input:

- Constraint Systems $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$.
- An index $i \in \mathbb{N}$.
- Starting values $z_0^{(1)} \in \mathbb{F}_1,\ z_0^{(2)} \in \mathbb{F}_2$.
- Claimed evaluations $z_i^{(1)} \in \mathbb{F}_1,\ z_i^{(2)} \in \mathbb{F}_2$
- An IVC Proof for iteration i $\pi_i := \left((\mathbbm{u}_i^{(1)}, \mathbbm{w}_i^{(1)}),\ (\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)}),\ (\mathbbm{u}_i^{(2)}, \mathbbm{w}_i^{(2)}),\ (\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})\right)$.

▶ Remark 10. The prior IVC proof π_i contained an additional instance-witness pair $(\mathbbm{u}_i^{(1)}, \mathbbm{w}_i^{(1)})$. This pair is no longer included in our modified verifier Section 5.2. As we explain in Section 7.2, the inclusion of these elements (along with misplaced checks) lead to the vulnerability.

The verifier performs an initial procedure:

1. Given functions F_1 and F_2, deterministically generate augmented R1CS constraint systems $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ which implement relations $\mathcal{R}_1$ and $\mathcal{R}_2$ from Figures 1a and 1b
2. Compute the folding verification key

$$(\,\cdot\,, \mathsf{vk}) \leftarrow \mathsf{Fold}_{\mathcal{K}}\big(\mathsf{pp},\ \mathsf{R1CS} := (\mathsf{R1CS}^{(1)}, \mathsf{R1CS}^{(2)})\big)$$

The verifier accepts if the following conditions are met:

1. The index i must be greater than 0.
2. $\mathbbm{u}_i^{(1)}.x_1 = \mathsf{H}_1\left(\mathsf{vk}, i^{(1)}, z_0^{(1)}, z_i^{(1)}, \mathbb{U}_i^{(2)}\right)$
3. $\mathbbm{u}_i^{(2)}.x_1 = \mathsf{H}_2\left(\mathsf{vk}, i^{(2)}, z_0^{(2)}, z_i^{(2)}, \mathbb{U}_i^{(1)}\right)$
4. Pair $(\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)})$ satisfies $\mathsf{R1CS}^{(1)}$.
5. Pairs $(\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})$ satisfies $\mathsf{R1CS}^{(2)}$.
6. Pair $(\mathbbm{u}_i^{(1)}, \mathbbm{w}_i^{(1)})$ strictly satisfies $\mathsf{R1CS}^{(1)}$.
7. Pair $(\mathbbm{u}_i^{(2)}, \mathbbm{w}_i^{(2)})$ strictly satisfies $\mathsf{R1CS}^{(2)}$.

7.2 The Vulnerability

In this section we first break down the implications of the verifier checks. Then, we explore a vulnerability with the approach. Finally, we describe a process to forge convincing IVC proofs in two stages.

Informally, for $i > 2$, the security argument for Nova IVC proceeds as follows:

- The verifier checks that $\mathbbm{u}_i^{(1)}.x_1 = \mathsf{H}_1\left(\mathsf{vk}, i^{(1)}, z_0^{(1)}, z_i^{(1)}, \mathbb{U}_i^{(2)}\right)$. This ensures that $\mathbbm{u}_i^{(1)}.x_1$ is derived from the inputs $z_i^{(1)}$ and $\mathbb{U}_i^{(2)}$ that are provided to the verifier.
- The verifier checks that the pair $(\mathbbm{u}_i^{(1)}, \mathbbm{w}_i^{(1)})$ satisfies $\mathsf{R1CS}^{(1)}$ which implements the relation $\mathcal{R}_1$. This implies two things:

  - First, $\mathbb{U}_i^{(2)}$ is the result of folding the instances $\mathbb{u}_{i-1}^{(2)}$ and $\mathbb{U}_{i-1}^{(2)}$ specified in $\mathbb{w}_i^{(1)}$,
  - Second, $\mathbb{u}_{i-1}^{(2)}.x_0 = \mathsf{H}_1\left(\mathsf{vk},\ (i-1)^{(1)},\ z_0^{(1)},\ z_{i-1}^{(1)},\ \mathbb{U}_{i-1}^{(2)}\right)$ where $z_i^{(1)} = \mathsf{F}_1(z_{i-1}^{(1)},\ \mathsf{aux}_{i-1}^{(1)})$ for some element $\mathsf{aux}_{i-1}^{(1)}$.
- The verifier checks that $(\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})$ satisfies $\mathsf{R1CS}^{(2)}$, which implements the relation $\mathcal{R}_2$. Then by knowledge soundness of the folding scheme, one can extract valid witnesses $\mathbb{w}_{i-1}^{(2)}$ for $\mathbb{u}_{i-1}^{(2)}$ and $\mathbb{W}_{i-1}^{(2)}$ for $\mathbb{U}_{i-1}^{(2)}$ with respect to $\mathsf{R1CS}^{(2)}$.
- Now, since $\mathbb{w}_{i-1}^{(2)}$ is a valid witness for $\mathbb{u}_{i-1}^{(2)}$, there are instances $\mathbb{u}_{i-1}^{(1)}$ and $\mathbb{U}_{i-2}^{(1)}$ specified in $\mathbb{w}_{i-1}^{(2)}$. By definition of $\mathcal{R}_2$, the instance $\mathbb{u}_{i-1}^{(1)}$ must satisfy $\mathbb{u}_{i-1}^{(1)}.x_1 = \mathbb{u}_{i-1}^{(2)}.x_0 = \mathsf{H}_1\left(\mathsf{vk}, (i-1)^{(1)}, z_0^{(1)}, z_{i-1}^{(1)}, \mathbb{U}_{i-1}^{(2)}\right)$.

We would now like to conclude that both $\mathbb{u}_{i-1}^{(1)}$ and $\mathbb{U}_{i-2}^{(1)}$ are satisfiable for $\mathsf{R1CS}^{(1)}$. However, none of the verifier checks or invariants induced by the relations imply that either $\mathbb{u}_{i-1}^{(1)}$ or $\mathbb{U}_{i-2}^{(1)}$ are satisfiable with respect to $\mathsf{R1CS}^{(1)}$. To see why, observe that $\mathcal{R}_2$ verifies that $\mathbb{u}_{i-1}^{(1)}$ and $\mathbb{U}_{i-2}^{(1)}$ fold into some $\mathbb{U}_{i-1}^{(1)}$. Then this $\mathbb{U}_{i-1}^{(1)}$ is hashed into $\mathbb{u}_{i-1}^{(2)}.x_1$, which gets copied to $\mathbb{u}_i^{(1)}.x_0$. The verifier is given an instance $\mathbb{U}_i^{(1)}$ that it expects to be the result of folding $\mathbb{u}_i^{(1)}$ and $\mathbb{U}_{i-1}^{(1)}$, but this need not be the case. In fact, $\mathbb{U}_i^{(1)}$ can be the result of folding entirely different $\mathbb{u}^{(1)}$ and $\mathbb{U}^{(1)}$.

Our attack exploits this by running the honest Nova prover for two stages. The first stage generates a satisfiable instance $\mathbb{u}_{i-1}^{(2)}$ with x_0 containing our own adversarially chosen values of $(i-1)^{(1)}$ and $z_{i-1}^{(1)}$. Then, the second stage generates pairs $(\mathbb{u}_i^{(1)}, \mathbb{w}_i^{(1)})$, $(\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})$ by running the honest prover again with $\mathbb{U}_\perp^{(2)}$, $\mathbb{u}_{i-1}^{(2)}$ as relational witness inputs. The attack proceeds symmetrically to generate pairs $(\mathbb{u}_i^{(2)}, \mathbb{w}_i^{(2)})$, $(\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)})$.

7.3 Attack Procedure

Our adversary $\mathcal{A}$ takes in as input:
- Constraint Systems $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$.
- An attack index $i > 2 \in \mathbb{N}$.
- Arbitrary starting values $z_0^{(1)} \in \mathbb{F}_1$ and $z_0^{(2)} \in \mathbb{F}_2$.
- Arbitrary claimed evaluations $z_i^{(1)} \in \mathbb{F}_1$ and $z_i^{(2)} \in \mathbb{F}_2$.
- Preimages $(z_{i-1}^{(1)},\ \mathsf{aux}_{i-1}^{(1)}) \in \mathbb{F}_1$ and $(z_{i-1}^{(2)},\ \mathsf{aux}_{i-1}^{(2)}) \in \mathbb{F}_2$ such that $z_i^{(1)} = \mathsf{F}_1(z_{i-1}^{(1)},\ \mathsf{aux}_{i-1}^{(1)})$ and $z_i^{(2)} = \mathsf{F}_2(z_{i-1}^{(2)},\ \mathsf{aux}_{i-1}^{(2)})$.

$\mathcal{A}$ will produce a false but convincing IVC proof π_i that the elements $z_i^{(1)} = \mathsf{F}_1^{(i)}(z_0^{(1)},\ \cdot)$ and $z_i^{(2)} = \mathsf{F}_2^{(i)}(z_0^{(2)},\ \cdot)$ are produced by iteratively applying the non-deterministic functions $\mathsf{F}_1,\ \mathsf{F}_2$ i-times on $z_0^{(1)},\ z_0^{(2)}$ for some collection of auxillary values $\{\mathsf{aux}_j^{(1)},\ \mathsf{aux}_j^{(2)}\}_{0 \le j < i}$.

Stage One. $\mathcal{A}$ will imitate an honest Nova Prover to produce a satisfying pair $(\mathbb{u}_{i-1}^{(2)}, \mathbb{w}_{i-1}^{(2)})$ for $\mathsf{R1CS}^{(2)}$, but with adversarial inputs.

1. **Produce Adversarial Instance:** We will produce an adversarial $\mathbb{u}_{i-1}^{(1)}$ by performing the following steps:
   a. Compute $x_0 := \mathsf{H}_2\left(\mathsf{vk}, (i-2)^{(2)}, z_0^{(2)}, z_{i-2}^{(2)}, \mathbb{U}_\perp^{(1)}\right)$, where $z_{i-2}^{(2)}$ can be set to anything, such as $\vec{0}^{(2)}$.
   b. Compute $x_1 := \mathsf{H}_1\left(\mathsf{vk}, (i-1)^{(1)}, z_0^{(1)}, z_{i-1}^{(1)}, \mathbb{U}_\perp^{(2)}\right)$.

c. Commit to the extended witness $\bar{\mathsf{w}}_{i-1}^{(1)} \leftarrow \mathsf{Commit}(\mathsf{pp}_W^{(1)}, w_{i-1}^{(1)})$, where the extended witness $w_{i-1}^{(1)}$ can be set to anything, such as $\vec{0}^{(2)}$.

d. Assign $\mathbb{u}_{i-1}^{(1)} := \left(\bar{0}^{(1)}, 1^{(1)}, \bar{\mathsf{w}}_{i-1}^{(1)}, (x_0, x_1)\right)$ and $\mathbb{w}_{i-1}^{(1)} := \left(\vec{0}^{(1)}, w_{i-1}^{(1)}\right)$.

2. **Fold Pair for $\mathsf{R1CS}^{(1)}$:** Fold the newly computed pair $(\mathbb{u}_{i-1}^{(1)}, \mathbb{w}_{i-1}^{(1)})$ with the trivially satisfiable pair $(\mathbb{U}_{\perp}^{(1)}, \mathbb{W}_{\perp}^{(1)})$ for $\mathsf{R1CS}^{(1)}$.

$$\mathsf{Fold}_{\mathcal{P}}\left(\mathsf{pk}, (\mathbb{u}_{i-1}^{(1)}, \mathbb{w}_{i-1}^{(1)}), (\mathbb{U}_{\perp}^{(1)}, \mathbb{W}_{\perp}^{(1)})\right) \rightarrow \left(\bar{\mathsf{T}}_{i-2}^{(1)}, (\mathbb{U}_{i-1}^{(1)}, \mathbb{W}_{i-1}^{(1)})\right)$$

Obtain a folding proof $\bar{\mathsf{T}}_{i-2}^{(1)}$ and new committed relaxed instance-witness pair $(\mathbb{U}_{i-1}^{(1)}, \mathbb{W}_{i-1}^{(1)})$.

3. **Compute New Pair for $\mathsf{R1CS}^{(2)}$:** Compute the new committed pair $(\mathbb{u}_{i-1}^{(2)}, \mathbb{w}_{i-1}^{(2)})$ for $\mathsf{R1CS}^{(2)}$ as follows:
   - Define $\hat{w}_{i-1}^{(2)} := (\mathsf{vk}, (i-2)^{(2)}, z_0^{(2)}, z_{i-2}^{(2)}, \mathsf{aux}_{i-2}^{(2)}, \mathbb{U}_{\perp}^{(1)}, \mathbb{u}_{i-1}^{(1)}, \bar{\mathsf{T}}_{i-2}^{(1)})$ as the relation witness for $\mathcal{R}_2$, where $\mathsf{aux}_{i-2}^{(2)}$ can be set to anything, such as $\vec{0}^{(2)}$. Then, compute the extended witness $w_{i-1}^{(2)}$ by performing the computation on $\hat{w}_{i-1}^{(2)}$ required to satisfy the constraints expressed in $\mathsf{R1CS}^{(2)}$.
   - Commit to the extended witness $\bar{\mathsf{w}}_{i-1}^{(2)} \leftarrow \mathsf{Commit}(\mathsf{pp}_W^{(2)}, w_{i-1}^{(2)})$.
   - Define $x_0 = \mathbb{u}_{i-1}^{(1)}.x_1$ and compute $x_1 = \mathsf{H}_2\left(\mathsf{vk}, (i-1)^{(2)}, z_0^{(2)}, \mathsf{F}_2^{(2)}(z_{i-2}^{(2)}, \mathsf{aux}_{i-2}^{(2)}), \mathbb{U}_{i-1}^{(1)}\right)$.
   - Assign $\mathbb{u}_{i-1}^{(2)} := \left(\bar{0}^{(2)}, 1^{(2)}, \bar{\mathsf{w}}_{i-1}^{(2)}, (x_0, x_1)\right)$ and $\mathbb{w}_{i-1}^{(2)} := \left(\vec{0}^{(2)}, w_{i-1}^{(2)}\right)$.

This new committed instance-witness pair $(\mathbb{u}_{i-1}^{(2)}, \mathbb{w}_{i-1}^{(2)})$ is valid, because the computation performed above explicitly satisfies the constraints of $\mathsf{R1CS}^{(2)}$. Furthermore, $\mathbb{u}_{i-1}^{(2)}.x_0 = \mathbb{u}_{i-1}^{(1)}.x_1$, which is maliciously set to $\mathsf{H}_1\left(\mathsf{vk}, (i-1)^{(1)}, z_0^{(1)}, z_{i-1}^{(1)}, \mathbb{U}_{\perp}^{(2)}\right)$.

Stage Two. $\mathcal{A}$ will imitate an honest Nova Prover for $\mathsf{R1CS}^{(1)}$, but with witness input derived in stage one.

1. **Fold Pair for $\mathsf{R1CS}^{(2)}$:** Fold the newly computed pair $(\mathbb{u}_{i-1}^{(2)}, \mathbb{w}_{i-1}^{(2)})$ with the trivially satisfiable pair $(\mathbb{U}_{\perp}^{(2)}, \mathbb{W}_{\perp}^{(2)})$ for $\mathsf{R1CS}^{(2)}$.

$$\mathsf{Fold}_{\mathcal{P}}\left(\mathsf{pk}, (\mathbb{u}_{i-1}^{(2)}, \mathbb{w}_{i-1}^{(2)}), (\mathbb{U}_{\perp}^{(2)}, \mathbb{W}_{\perp}^{(2)})\right) \rightarrow \left(\bar{\mathsf{T}}_{i-1}^{(2)}, (\mathbb{U}_{i}^{(2)}, \mathbb{W}_{i}^{(2)})\right)$$

Obtain a folding proof $\bar{\mathsf{T}}_{i-1}^{(2)}$ and new committed relaxed instance-witness pair $(\mathbb{U}_{i}^{(2)}, \mathbb{W}_{i}^{(2)})$. Note that since both pairs are satisfiable, this new pair $(\mathbb{U}_{i}^{(2)}, \mathbb{W}_{i}^{(2)})$ is also satisfiable.

2. **Compute New Pair for $\mathsf{R1CS}^{(1)}$:** Compute the new committed pair $(\mathbb{u}_{i}^{(1)}, \mathbb{w}_{i}^{(1)})$ for $\mathsf{R1CS}^{(1)}$ as follows:
   - Define $\hat{w}_{i}^{(1)} := (\mathsf{vk}, (i-1)^{(1)}, z_0^{(1)}, z_{i-1}^{(1)}, \mathsf{aux}_{i-1}^{(1)}, \mathbb{U}_{\perp}^{(2)}, \mathbb{u}_{i-1}^{(2)}, \bar{\mathsf{T}}_{i-1}^{(2)})$ as the relation witness for $\mathcal{R}_1$. Then, compute the extended witness $w_i^{(1)}$ by performing the computation on $\hat{w}_i^{(1)}$ required to satisfy the constraints expressed in $\mathsf{R1CS}^{(1)}$.
   - Commit to the extended witness $\bar{\mathsf{w}}_{i}^{(1)} \leftarrow \mathsf{Commit}(\mathsf{pp}_W^{(1)}, w_{i}^{(1)})$.
   - Define $x_0 = \mathbb{u}_{i-1}^{(2)}.x_1$ and compute $x_1 = \mathsf{H}_1\left(\mathsf{vk}, i^{(1)}, z_0^{(1)}, z_i^{(1)} := \mathsf{F}_1(z_{i-1}^{(1)}, \mathsf{aux}_{i-1}^{(1)}), \mathbb{U}_{i}^{(2)}\right)$.
   - Assign $\mathbb{u}_{i}^{(1)} := \left(\bar{0}^{(1)}, 1^{(1)}, \bar{\mathsf{w}}_{i}^{(1)}, (x_0, x_1)\right)$ and $\mathbb{w}_{i}^{(1)} := \left(\vec{0}^{(1)}, w_{i}^{(1)}\right)$.

This new committed instance-witness pair $(\mathbb{u}_i^{(1)}, \mathbb{w}_i^{(1)})$ is satisfiable, because the computation performed above explicitly satisfies the constraints of $\mathsf{R1CS}^{(1)}$. Furthermore, $\mathbb{u}_i^{(1)}.x_1 = \mathsf{H}_1\left(\mathsf{vk}, i^{(1)}, z_0^{(1)}, z_i^{(1)}, \mathbb{U}_i^{(2)}\right)$. To recap, after these two stages we obtained pairs $(\mathbb{u}_i^{(1)}, \mathbb{w}_i^{(1)})$ and $(\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})$ such that the following hold:

- $\mathbb{u}_i^{(1)}.x_1 = \mathsf{H}_1\left(\mathsf{vk}, i^{(1)}, z_0^{(1)}, z_i^{(1)}, \mathbb{U}_i^{(2)}\right)$
- $(\mathbb{u}_i^{(1)}, \mathbb{w}_i^{(1)})$ satisfies $\mathsf{R1CS}^{(1)}$.
- $(\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})$ satisfies $\mathsf{R1CS}^{(2)}$.

Symmetry. Since the relations expressed by the augmented constraint systems $\mathsf{R1CS}^{(1)}$ and $\mathsf{R1CS}^{(2)}$ are symmetric (Section 4) when $i > 2$. We can repeat both stages above symmetrically to produce pairs $(\mathbb{u}_i^{(2)}, \mathbb{w}_i^{(2)})$ and $(\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)})$ such that the following hold:

- $\mathbb{u}_i^{(2)}.x_1 = \mathsf{H}_2\left(\mathsf{vk}, i^{(2)}, z_0^{(2)}, z_i^{(2)}, \mathbb{U}_i^{(1)}\right)$
- $(\mathbb{u}_i^{(2)}, \mathbb{w}_i^{(2)})$ satisfies $\mathsf{R1CS}^{(2)}$.
- $(\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)})$ satisfies $\mathsf{R1CS}^{(1)}$.

Finally, adversary $\mathcal{A}$ outputs an IVC proof $\pi_i := \left((\mathbb{u}_i^{(1)}, \mathbb{w}_i^{(1)}),\ (\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)}),\ (\mathbb{u}_i^{(2)}, \mathbb{w}_i^{(2)}),\ (\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})\right)$. By construction, π_i is a convincing IVC proof (i.e. all the verifier checks pass).

7.3.1 Proof of Concept Attack Against the Minroot Verifier

We implement our attack against the prior 2-cycle Nova Proof System generically for any choice of F_1 and F_2 and parameters specified in Section 7.3. Our implementation can be found in our repo MercysJest/NovaBreakingTheCycleAttack, which is a direct fork of the original microsoft/Nova repo. To demonstrate the attack, we can compute a convincing Nova proof for the correct evaluation of 2^{75} rounds of the Minroot VDF in only 116 milliseconds on a Macbook. The demonstration code can be found in examples/vuln.rs.

```
=========================================================================
Demonstrating exploit against Nova-based VDF with MinRoot delay function
=========================================================================
Producing public parameters...
PublicParams::setup, took 2.9136875s
...
Each IVC Step Performs 4096 iterations of Minroot.
Generating fake proof of 9223372036854775808 IVC Steps.
In total, faking 37778931862957161709568 Minroot iterations.
Generating fake proof took 115.872416ms
Verifying a RecursiveSNARK...
RecursiveSNARK::verify: true, took 27.8225ms
Generating a CompressedSNARK using Spartan with IPA-PC...
CompressedSNARK::prove: true, took 1.5859465s
CompressedSNARK::len 9713 bytes
Verifying a CompressedSNARK...
CompressedSNARK::verify: true, took 55.04425ms
```

8 Malleability of Nova's IVC proofs

In this section, we show that the 2-cycle Nova IVC proofs, described in Section 5 are malleable. This attack readily generalizes to the original (single chain) Nova construction [12]. We later discuss how to prevent this malleability attack by making use of either an additional ctx element in the verification key vk or use of a simulation-extractable zkSNARK (e.g., Spartan) for IVC proof compression.

Suppose an adversary is given a valid Nova IVC proof π_i with respect to the following parameters

- IVC public parameters pp,
- a description of functions $\mathsf{F}_1 : \mathbb{F}_1^{a_1} \times \mathbb{F}_1^{b_1} \to \mathbb{F}_1^{a_1}$ and $\mathsf{F}_2 : \mathbb{F}_2^{a_2} \times \mathbb{F}_2^{b_2} \to \mathbb{F}_2^{a_2}$,
- an index $i \in \mathbb{N}$,
- starting values $z_0^{(1)} \in \mathbb{F}_1^{a_1}$ and $z_0^{(2)} \in \mathbb{F}_2^{a_2}$, and
- claimed evaluations $z_i^{(1)} \in \mathbb{F}_1^{a_1}$ and $z_i^{(2)} \in \mathbb{F}_2^{a_2}$.

We show in Section 8.1 that the adversary can construct a proof π_{prime} for the same iteration i, but for some $z_{\mathsf{prime}}^{(2)}$ different from $z_i^{(2)}$. In particular, running the IVC verifier with arguments

$$\left(\mathsf{pp},\ (\mathsf{F}_1, \mathsf{F}_2),\ i,\ (z_0^{(1)}, z_0^{(2)}),\ (z_i^{(1)}, z_{\mathsf{prime}}^{(2)}),\ \pi_{\mathsf{prime}}\ \right)$$

causes the verifier to accept. We stress that our adversary does not need to know the auxiliary values $(\mathsf{aux}_0^{(2)}, \mathsf{aux}_1^{(2)}, \ldots, \mathsf{aux}_{i-1}^{(2)})$ used to compute $z_i^{(2)}$. By choosing an alternate final auxiliary value $\mathsf{aux}_{\mathsf{prime}}^{(2)} \neq \mathsf{aux}_{i-1}^{(2)}$, our adversary can construct a proof π_{prime} for an alternate value $z_{\mathsf{prime}}^{(2)}$ for i iterations, without knowledge of the first $i-1$ auxiliary values. We discuss two ways to mitigate this issue in Section 8.2 below.

Why does this matter? A malleable proof system [5] can lead to a real world security vulnerability. Suppose Alice uses her secret auxiliary values to compute $z_i^{(2)}$ and this $z_i^{(2)}$ encodes her payment address. She sends the $z_i^{(2)}$ and the proof to a payment contract. An attacker could intercept her message and maul $z_i^{(2)}$ to a $z_{\mathsf{prime}}^{(2)}$ which encodes the attackers payment address instead, along with a valid proof π_{prime}. The payment contract will then send the funds to the attacker instead of Alice. Concretely, if Tornado Cash had used a proof system that were malleable on statements, it would have been possible to steal funds.

8.1 The Malleability Attack

We present a malleability attack on the last step of the IVC chain. Recall that the Nova IVC proof π_i contains the following elements

$$\pi_i := \left((\mathbbm{u}_i^{(2)}, \mathbbm{w}_i^{(2)}),\ (\mathbb{U}_i^{(1)}, \mathbb{W}_i^{(1)}),\ (\mathbb{U}_i^{(2)}, \mathbb{W}_i^{(2)})\right).$$

The malleability attack proceeds as follows:

1. **Parse witness:** In Section 6, we argued that we can parse the witness $\mathbbm{w}_i^{(2)}$ to obtain relational witness

   $$\hat{w}_i^{(2)} = \left(\mathsf{vk},\ (i-1)^{(2)},\ z_0^{(2)}, z_{i-1}^{(2)},\ \mathsf{aux}_{i-1}^{(2)},\ \mathbb{U}_{i-1}^{(1)}, \mathbbm{u}_i^{(1)},\ \bar{\mathsf{T}}_{i-1}^{(1)}\right)$$

   for which we know

   $$z_i^{(2)} = \mathsf{F}_2\left(z_{i-1}^{(2)}, \mathsf{aux}_{i-1}^{(2)}\right)$$
   $$\mathbb{U}_i^{(1)} = \mathsf{Fold}_{\mathcal{V}}\left(\mathsf{vk},\ \mathbb{U}_{i-1}^{(1)},\ \mathbbm{u}_i^{(1)},\ \bar{\mathsf{T}}_{i-1}^{(1)}\right)$$

   Thus, we parse $\mathbbm{w}_i^{(2)}$ to obtain $\hat{w}_i^{(2)}$
2. **Find a different auxiliary value:** Using $z_{i-1}^{(2)}$, choose some $\mathsf{aux}_{\mathsf{prime}}^{(2)}$ such that

   $$z_{\mathsf{prime}}^{(2)} := \mathsf{F}_2\left(z_{i-1}^{(2)},\ \mathsf{aux}_{\mathsf{prime}}^{(2)}\right) \quad \neq \quad \mathsf{F}_2\left(z_{i-1}^{(2)},\ \mathsf{aux}_{i-1}^{(2)}\right) = z_i^{(2)}$$

   We assume that finding such an $\mathsf{aux}_{\mathsf{prime}}^{(2)}$ is efficient for F_2.

3. **Compute a new pair for $\mathsf{R1CS}^{(2)}$:** Compute the pair $(\mathbb{u}^{(2)}_{\mathsf{prime}}, \mathbb{w}^{(2)}_{\mathsf{prime}})$ for $\mathsf{R1CS}^{(2)}$ as follows:
   - Define $\hat{w}^{(2)}_{\mathsf{prime}} := (\mathsf{vk}, (i-1)^{(2)}, z^{(2)}_0, z^{(2)}_{i-1}, \mathsf{aux}^{(2)}_{\mathsf{prime}}, \mathbb{U}^{(1)}_{i-1}, \mathbb{u}^{(1)}_i, \bar{\mathsf{T}}^{(1)}_{i-i})$ as the relation witness for $\mathcal{R}_2$. Then, compute the extended witness $w^{(2)}_{\mathsf{prime}}$ by performing the computation on $\hat{w}^{(2)}_{\mathsf{prime}}$ required to satisfy the constraints expressed in $\mathsf{R1CS}^{(2)}$.
   - Commit to the extended witness $\bar{\mathsf{w}}^{(2)}_{\mathsf{prime}} \leftarrow \mathsf{Commit}(\mathrm{pp}^{(2)}_W, w^{(2)}_{\mathsf{prime}})$.
   - Define $x_0 := \mathbb{u}^{(1)}_i.x_1$ and $x_1 := \mathsf{H}_2\big(\mathsf{vk}, i^{(2)}, z^{(2)}_0, z^{(2)}_{\mathsf{prime}} := \mathsf{F}_2(z^{(2)}_{i-1}, \mathsf{aux}^{(2)}_{\mathsf{prime}}), \mathbb{U}^{(1)}_i\big)$.
   - Assign $\mathbb{u}^{(2)}_{\mathsf{prime}} := \big(\bar{0}^{(2)}, 1^{(2)}, \bar{\mathsf{w}}^{(2)}_{\mathsf{prime}}, (x_0, x_1)\big)$ and $\mathbb{w}^{(2)}_{\mathsf{prime}} := \big(\vec{0}^{(2)}, w^{(2)}_{\mathsf{prime}}\big)$.
4. **Output mauled proof:** Output $\pi_{\mathsf{prime}} := \big((\mathbb{u}^{(2)}_{\mathsf{prime}}, \mathbb{w}^{(2)}_{\mathsf{prime}}),\ (\mathbb{U}^{(1)}_i, \mathbb{W}^{(1)}_i),\ (\mathbb{U}^{(2)}_i, \mathbb{W}^{(2)}_i)\big)$.

By construction, the proof π_{prime} is convincing.

▶ Remark 11 (Generalizing the Malleability Attack). In more general terms, given an IVC proof that contains information about a valid pre-image z_{i-1} to z_i Our malleability attack re-executes the last step of the IVC prover with a different choice of the final auxiliary value aux_{i-1}. In particular, our attack readily generalizes to the original Nova construction [12].

8.2 Preventing This Malleability Attack

There are several strategies that defeat this specific malleability attack.

Incorporating Context Elements. The first approach is to expand the verification key vk to include a context string ctx which includes the IVC verifier context

$$(\,\cdot\,, \mathsf{vk}) \leftarrow \mathsf{Fold}_{\mathcal{K}}\big(\mathsf{pp},\ \mathsf{R1CS} := (\mathsf{R1CS}^{(1)}, \mathsf{R1CS}^{(2)}),\ \mathsf{ctx} := (i, z^{(1)}_i, z^{(2)}_i)\big)$$

This inductively binds the proof to a particular choice of $(i, z^{(1)}_i, z^{(2)}_i)$. However, this breaks the incremental property (namely, completeness after iteration i as described in Definition 1) of the proof since the prover cannot use this proof to produce another valid proof for an iteration $j > i$. Additionally, the IVC prover must compute the evaluations $(z^{(1)}_i, z^{(2)}_i)$ before generating the IVC proof.

Compression. A different defense is to use a compressed IVC proof π'_i, namely

$$\pi'_i := \big(\mathbb{u}^{(2)}_i,\ \mathbb{U}^{(2)}_i,\ \mathbb{U}^{(1)}_i,\ \bar{\mathsf{T}}^{(2)}_i,\ \pi_{\mathsf{sat}}\big) \tag{2}$$

where π_{sat} is a SNARK proof for the relation

$$\mathcal{R}_{\mathsf{sat}} := \left\{ \big(\mathbb{U}^{(2)}_{i+1},\ \mathbb{U}^{(1)}_i\ ;\ \mathbb{W}^{(2)}_{i+1},\ \mathbb{W}^{(1)}_i\big) \quad : \quad \begin{array}{c} (\mathbb{U}^{(1)}_i, \mathbb{W}^{(1)}_i) \text{ satisfies } \mathsf{R1CS}^{(1)} \\ \wedge\ (\mathbb{U}^{(2)}_{i+1}, \mathbb{W}^{(2)}_{i+1}) \text{ satisfies } \mathsf{R1CS}^{(2)} \end{array} \right\} \tag{3}$$

Here the SNARK must be zero knowledge so that π_{sat} contains no information about the underlying witnesses. Similarly, the witness commitment $\bar{\mathsf{w}}^{(2)}_i$ in $\mathbb{u}^{(2)}_i$ must be a hiding commitment. Furthermore, the SNARK may also need to be simulation extractable [9]. The Spartan SNARK [18] is simulation-extractable [3]. Unfortunately, applying the SNARK compression step would remove the efficient incremental property of the IVC since we can no longer run the native Nova IVC prover for subsequent iterations. Nova [12, 14] uses Spartan produce compressed IVC proofs π'_i (2).

References

1 Benedikt Bünz and Binyi Chen. ProtoStar: Generic efficient accumulation/folding for special sound protocols. Cryptology ePrint Archive, Paper 2023/620, 2023. URL: `https://eprint.iacr.org/2023/620`.

2 Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 681–710, Virtual Event, August 16–20 2021. Springer, Heidelberg, Germany. `doi:10.1007/978-3-030-84242-0_24`.

3 Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 531–562, Lyon, France, April 23–27 2023. Springer, Heidelberg, Germany. `doi:10.1007/978-3-031-30617-4_18`.

4 Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. Weak fiat-shamir attacks on modern proof systems. Cryptology ePrint Archive, Paper 2023/691, 2023. URL: `https://eprint.iacr.org/2023/691`.

5 Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 566–598, Santa Barbara, CA, USA, August 19–23 2001. Springer, Heidelberg, Germany. `doi:10.1007/3-540-44647-8_33`.

6 Morris Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions, 2015-08-04 2015. `doi:10.6028/NIST.FIPS.202`.

7 Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. `doi:10.1007/3-540-47721-7_12`.

8 Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 519–535. USENIX Association, August 11–13 2021.

9 Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459, Shanghai, China, December 3–7 2006. Springer, Heidelberg, Germany. `doi:10.1007/11935230_29`.

10 Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. MinRoot: Candidate sequential function for Ethereum VDF. Cryptology ePrint Archive, Paper 2022/1626, 2022. URL: `https://eprint.iacr.org/2022/1626`.

11 Abhiram Kothapalli and Srinath Setty. HyperNova: Recursive arguments for customizable constraint systems. Cryptology ePrint Archive, Paper 2023/573, 2023. URL: `https://eprint.iacr.org/2023/573`.

12 Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388, Santa Barbara, CA, USA, August 15–18 2022. Springer, Heidelberg, Germany. `doi:10.1007/978-3-031-15985-5_13`.

13 Nicholas Mohnblatt. Sangria: A folding scheme for PLONK, 2023. link.

14 Nova Contributors. Nova implementation, 2022. URL: `https://github.com/Microsoft/Nova`.

15 oskarth. Towards a nova-based ZK virtual machine. `https://zkresear.ch/t/towards-a-nova-based-zk-vm/105`, 2023.

16 Pasta Contributors. Pasta curves, 2020. URL: `https://github.com/zcash/pasta_curves`.

17 Carla Ràfols and Alexandros Zacharakis. Folding schemes with selective verification. Cryptology ePrint Archive, Report 2022/1576, 2022. URL: https://eprint.iacr.org/2022/1576.

18 Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737, Santa Barbara, CA, USA, August 17–21 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-56877-1_25.

19 Srinath Setty. Nova pull request 167, 2023. URL: https://github.com/Microsoft/Nova/pull/167.

20 Supernational. Open VDF: Accelerating the nova snark-based vdf. https://medium.com/supranational/open-vdf-accelerating-the-nova-snark-based-vdf-2d00737029bd, 2023.

21 Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, March 19–21 2008. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-78524-8_1.

Censorship Resistance in On-Chain Auctions

Elijah Fox
Duality Labs, New York, NY, USA, USA

Mallesh M. Pai
Department of Economics, Rice University, Houston, TX, USA
Special Mechanisms Group

Max Resnick
Special Mechanisms Group

Abstract

Modern blockchains guarantee that submitted transactions will be included eventually; a property formally known as liveness. But financial activity requires transactions to be included in a timely manner. Classical liveness does *not* guarantee this, particularly in the presence of a motivated adversary who benefits from censoring transactions. We define *censorship resistance* as the amount it would cost the adversary to censor a transaction for a fixed interval of time as a function of the associated tip. This definition has two advantages, first it captures the fact that transactions with a higher miner tip can be more costly to censor, and therefore are more likely to swiftly make their way onto the chain. Second, it applies to a finite time window, so it can be used to assess whether a blockchain is capable of hosting financial activity that relies on timely inclusion.

We apply this definition in the context of auctions. Auctions are a building block for many financial applications, and censoring competing bids offers an easy-to-model motivation for our adversary. Traditional proof-of-stake blockchains have poor enough censorship resistance that it is difficult to retain the integrity of an auction when bids can only be submitted in a single block. As the number of bidders n in a single block auction increases, the probability that the winner is not the adversary, and the economic efficiency of the auction, both decrease faster than $1/n$. Running the auction over multiple blocks, each with a different proposer, alleviates the problem only if the number of blocks grows faster than the number of bidders. We argue that blockchains with more than one concurrent proposer can have strong censorship resistance. We achieve this by setting up a prisoner's dilemma among the proposers using conditional tips.

2012 ACM Subject Classification Applied computing → Online auctions

Keywords and phrases Censorship Resistance, Auctions, Blockchain, MEV

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.19

Related Version A full version of this paper including omitted proofs is available on ArXiv.
Full Version: `https://arxiv.org/abs/2301.13321`

1 Introduction

Blockchain consensus algorithms typically guarantee *liveness*, meaning valid transactions will be included on chain eventually. But financial applications, are time sensitive. For these to function as intended, valid transactions must be included on the blockchain in a timely manner. This requires something stronger than liveness: *censorship resistance*. To quote [5], censorship resistance is "ensuring that transactions that people want to put into the blockchain will actually get in in a timely fashion, even if "the powers that be", at least on that particular blockchain, would prefer otherwise."

In this paper, we propose a formal definition that quantifies censorship resistance in the sense of [5] above. We abstract away from the details of the chain and view it as a public bulletin board with two operations: a read operation, which always succeeds, and a write operation. The write operation succeeds when the transaction with an associated tip is added to the bulletin board and fails otherwise. We then define the *censorship resistance* of this public bulletin board as the amount it would cost a motivated adversary to cause a write operation to fail, as a function of the associated tip.

This definition has two advantages that are useful in applications. The first stems from the fact that we define censorship resistance as a function of the underlying tip: this captures how transactions with a higher tip can be more expensive to censor, and therefore are more likely to successfully make it on chain. The second is that by capturing the censorship resistance of a specific public bulletin board, with an associated length of time for a transaction to be added, we can capture the trade-off between censorship resistance and speed for specific blockchain designs.

Having presented this definition, we apply it to tackle whether a given public bulletin board is sufficiently censorship resistant to host a given mechanism. This is tricky because it depends on both the tips of the underlying transaction(s) *and* the motivation of the adversary, both of which are potentially endogenously determined by the mechanism.

This paper considers whether existing proof-of-stake blockchains are sufficiently censorship resistant to host time-sensitive auctions. We consider auctions for two reasons. First, the cost of being censored and the benefits of censoring competing bids are easy to quantify in an auction. Therefore, for a given public bulletin board with fixed censorship resistance, we can determine whether the auction will actually function as intended. Second, auctions are already a popular mechanism on-chain: for example, Maker DAO, the entity behind popular stablecoin DAI, uses Dutch clock auctions to sell the right to liquidate collateral for distressed loans, and additionally digital goods such as NFTs may also be auctioned off on-chain. Also, several important future developments will require auctions on-chain, from the very organization of Ethereum ([22]), to proposals by other large organizations to move current off-chain auctions on-chain ([12]).

Formally, we consider a seller of a single unit of an indivisible good who runs a second-price auction on-chain. The seller encodes the rules of the second-price auction in a smart contract. The contract selects the bidder who submitted the highest bid over a predefined period and sets the payment to the second-highest bid. But since all of this takes place on a blockchain, before a bid can be submitted to the auction, it must be included in a transaction on-chain. Valid transactions are submitted to a *mempool*. Each slot, the proposer gathers transactions from the mempool into a block that will eventually be added to the chain. Proposers have complete autonomy over which transactions to include.

The power of the proposer to determine the contents of the block and, therefore, the outcome of the auction sets up a competition for inclusion. Bidders include tips for the proposer along with their bids. The proposer receives these tips if and only if the corresponding transactions are included in his block. We suppose that there is a single colluding bidder who may offer a bribe to the proposer in exchange for omitting certain transactions. In distributed systems terminology, this single colluding bidder who may bribe the proposer if they believe it profitable for them to do so is our *threat model*.

We show that in the auction setting, tips for inclusion are a public good since they provide security to other transactions and only benefit the bidder who pays the tip if they win the auction. In contrast, bribes for omission are purely for private benefit and make other bidders worse off in equilibrium. Consequently, our results suggest that the colluding bidder is highly advantaged in this game. In particular, we show that as the number of honest bidders increases, the colluding bidder wins the auction increasingly often and collects an increasingly large share of the surplus created by the auction.

Notably, we assume that bids are sealed, that is, that the colluding bidder must choose which transactions to try and omit based solely on the associated tip. We show that the colluding bidder can back out the private bids (and therefore whether it is profitable to attempt to censor these bids) based on these public tips. This suggests that cryptographic approaches (e.g. commit-reveal schemes, encrypted mempools) are not a silver bullet for resolving censorship concerns in such settings.

We then consider two alternate designs that improve censorship resistance. The first is to run the auction over multiple slots with a different proposer for each. We find that this achieves sufficient censorship resistance only if the number of blocks grows faster than the number of bidders. This is undesirable for reasons external to our model; for example, executing the auction in a short window is important for financial applications. MEV auctions in particular are concerned about speed, since they need to clear at least once every slot – once every 12 seconds on Ethereum.

The second is to have blockchains with multiple concurrent block proposers, $k > 1$, and allow bidder tips to be conditioned not only on inclusion, but also on the number of proposers who include the bid within a slot. This allows bidders to set up a sort of "prisoner's dilemma" among the proposers by offering to pay a large tip T when only one proposer includes, and a small tip $t \ll T$ if multiple proposers include. Each proposer is incentivized to include since if they are alone in including the transaction, there is a high tip attached. Therefore, all proposers include the bid in equilibrium. However, censoring is expensive, since censoring requires that each proposer be bribed T for a total cost of kT. This leads to a low expected tip of kt but an asymmetrically expensive censorship cost of $kT \gg kt$. This asymmetry allows for a pooling equilibrium in which the probability of censorship is 0, the tips no longer reveal the bids, and the expected total tips are low.

2 Related Literature

Censorship resistance, for various definitions of the term, is a key desideratum motivating the adoption of blockchains. This property appeared in some of the earliest writings on the subject, e.g., [5]. More recently, this property has come under additional scrutiny due to two major developments. The first, Proposer Builder Separation (PBS) in Ethereum, explicitly establishes an auction for the right to build the next block. Block builders who win this auction decide which transactions make it onto the chain and, more importantly for our purpose, which transactions do not. PBS therefore enables a motivated adversary to censor specific transaction(s) by purchasing the right to build the next block and intentionally omitting those transactions ([6]). The second relates to US OFAC sanctions on certain Ethereum addresses, and the subsequent decision by certain block builders to exclude transactions including those addresses from blocks that they build. Effects of this (and a related definition of censorship resistance) are studied in [26].

The literature on auctions, even restricting to papers that explicitly think about auctions in the context of blockchains, is much larger. In such auctions, bids are rarely announced simultaneously, and maintaining the seal on bids transmitted through public channels requires cryptography. For example, a simple cryptographic second-price sealed-bid auction involves bidders submitting the hash of their bids rather than the bids themselves and then revealing the hash after all bids have been submitted. [11] showed that, using a cryptographically secure commitment scheme, it is possible to design an auction that is optimal, strategy proof and credible (in the sense of [1, 2]). More complicated cryptographic approaches can eliminate the need to reveal any information beyond the results of the auction and can also accommodate combinatorial auctions ([20, 9, 24]).

Auctions are commonly cited as use cases for the verifiable computation that smart contracts provide. This was originally envisaged in [25], who noted that "... a blockchain with a built-in fully fledged Turing-complete programming language that can be used to create "contracts" ... simply by writing up the logic in a few lines of code", see also [14, 3]. Auctions have also been suggested as a desirable mechanism to decide the order and inclusion of transactions to mitigate MEV ([18]). Historically, these were decided by a combination of auction and speed-based mechanisms, leading [8] to compare MEV with high-frequency trading as described in [4]. Initially, inclusion and priority within the block were decided by priority gas auctions (PGAs), since most validators gathered transactions directly from the mempool and ordered them according to their miner tips, breaking ties using a first come first serve rule ([8]). But recently, a super-majority of validators have switched their execution clients to MEV-boost compatible versions, meaning the right to decide inclusion and ordering for most blocks is sold to the highest bidder. These bidders are typically established builders who specialize in extracting the maximum value from each block. The leading advocate for this approach has been Flashbots, the company behind the initial open source MEV client. Their next product SUAVE, aims to move these auctions on-chain [12].

Previous MEV mitigation research has focused on fairness rather than censorship ([17, 16]). But [10] showed that for every sequencing rule of trades through a liquidity pool, there exists a way for the proposer to obtain non-zero risk-free profits suggesting that ordering based MEV is inevitable with current on chain financial application design. In response to this, researchers have suggested that frequent batch auctions or other order-agnostic mechanisms might alleviate the MEV that arises from transaction ordering power ([15]).

On-chain auctions have also been studied as a mechanism for the sale of non-fungible tokens (NFTs) [21]. Gradual dutch auctions (GDAs) [13], are a dynamic mechanism for selling multiple NFTs. [19] explores the credibility of GDAs and finds that an auctioneer can bid to artificially raise the sale price and create the appearance of demand.

3 Formalizing Censorship Resistance

As we described above, we abstract away from the details of the blockchain and instead consider solely the functionality as a public bulletin board. The public bulletin board can be written to, which is how bids may be submitted, and can be read from, which is how the auction can then be executed. For simplicity, we assume that once a message has been successfully written to the bulletin board, it can be read without friction. For example, any transaction included in an Ethereum block can be read by any full node.

▶ **Definition 1** (Public Bulletin Board). *A Public Bulletin Board has two public functions:*

1. $\text{write}(m, t)$ *takes as input a message m and an inclusion tip t and returns* 1 *if the message is successfully written to the bulletin board and* 0 *otherwise.*
2. $\text{read}()$ *returns a list of all messages that have been written to the bulletin board over the period.*

Some subtlety must be observed in the definition of a public bulletin board. First, the write function takes as input not only a message m but also a tip t. Here our definition departs from [7]. This models proposer tips and other forms of validator bribes. To motivate this, consider that the write function on Ethereum is unlikely to succeed without a sufficient tip, and so the behavior of write is very different depending on the size of the associated tip.

▶ **Example 2** (Single Block). In the case of a single block, the write(m, t) operation consists of submitting a transaction m with associated tip t. write(m, t) succeeds if the transaction is included in that block on the canonical chain.

▶ **Example 3** (Multiple Blocks). In the case of k blocks with rotating proposers, the write(m, t) operation consists of submitting a transaction m with associated tip t during the period before the first block is formed. write(m, t) succeeds if the transaction is included in any of the k blocks.

We can now model the relationship between the tip t and the success of the write operation as a function. This function provides a flexible definition of the bulletin board's censorship resistance.

▶ **Definition 4** (Censorship Resistance of a Public Bulletin Board). *The censorship resistance of a public bulletin board $\mathcal{D}$ is a mapping $\phi : \mathbb{R}_+ \to \mathbb{R}_+$ that takes as input the tip t corresponding to the tip in the write operation* write$(\cdot, t)$ *and outputs the minimum cost that a motivated adversary would have to pay to make the* write *fail.*

This definition allows us to compare the censorship resistance of two bulletin boards even when the inner machinations of those mechanisms are profoundly different. This definition also easily extends to cases where tips are multi-dimensional. i.e. $t \in \mathbb{R}^n_+$ by substituting $\phi : \mathbb{R}_+ \to \mathbb{R}_+$ to $\phi : \mathbb{R}^n_+ \to \mathbb{R}_+$.

▶ **Example 2** (continued). The cost to censor this transaction would simply be t in the uncongested case because the motivated adversary has to compensate the proposer at least as much as the proposer would be losing from the tip. So:

$$\phi(t) = t \tag{1}$$

As a variant, suppose that the transaction fee mechanism involves burning some portion of the tip as is the case on Ethereum today after the EIP 1559 upgrade [23]. Suppose further that the current burn is b, i.e., with a tip of t only $t - b$ is paid to the proposer upon inclusion and the rest is burned. In this case, the proposer would be willing to censor upon a bribe of at least $t - b$, so:

$$\phi(t) = \max(t - b, 0) \tag{2}$$

▶ **Example 3** (continued). In the case of k blocks with rotating proposers, the write(m, t) operation consists of submitting a transaction m with associated tip t during the period before the first block is formed. write(m, t) succeeds if the transaction is included in any of the k blocks. The cost to censor this transaction would be kt since each of the k proposers must be bribed at least t to compensate them for the forgone tip on the transaction that they each had the opportunity to include. So:

$$\phi(t) = kt. \tag{3}$$

4 Modelling the Auction

We are now in a position to use our definitions to understand the censorship resistance of a sealed-bid auction, for various design parameters.

We consider a traditional independent private values setting. There is a single seller with a single unit of an indivisible good for sale. There are $n + 1$ buyers for the good, $n \geq 1$ – we denote the set of bidders by $N = \{0, 1, \ldots, n\}$. Each of these buyers $i \in N$ has a private value for the good, v_i . We suppose buyer 0's value v_0 is drawn from a distribution with CDF F_0 and density f_0, and the other buyers' values are are drawn i.i.d. from a distribution with CDF F and density f. Both distributions have bounded support, we normalize these to be equal to the unit interval $[0, 1]$. Several of our results will be for the special case $F = F_0 = U[0, 1]$. Bidders know their own v_i; and n, F, and F_0 are common knowledge among all bidders and the seller.

The seller wants to conduct a sealed bid second-price auction with reserve price r. As described in this introduction, the point of departure of our model is that this auction runs on a blockchain. Initially, we consider an auction that accepts bids in a single designated block. Below, we formally define this game and our solution concept.

In an idealized world with honest/ non-strategic proposers, the auction would run as follows:

1. The seller announces the auction.
2. All buyers privately commit their bids to the auction as transactions.
3. Proposer(s) include these transactions on relevant block(s).[1]
4. The second-price auction is computed based on the included bids, i.e., the highest bid is selected to win if this bid $\geq r$, in this case paying a price of $\max\{r, \text{other bids}\}$.

In particular, we assume that the idealized sealed-bid nature of off-chain auctions can be achieved on-chain via cryptographic methods[2]. We also assume that the set of bids submitted for the auction is public (the bid itself may be private, but the fact that it exists as a bid is public).

Our main concern is that bids submitted for this auction may be censored, that is, omitted from a block. More specifically, we suppose that after all other bids are submitted, but before they are revealed, a designated bidder, bidder 0, can pay the proposer of the block to *censor* bids. These censored bids are then excluded from the block and have no impact on the auction. We assume that the proposer is purely profit focused and that bidder's utilities are quasilinear.

Formally, we consider the following game:

1. The seller announces second-price sealed-bid auction with reserve price r to be conducted over a single block.
2. Buyers learn their values $v_i \leftarrow F$.
3. Buyers $1, \ldots, n$ each simultaneously submit a private bid b_i and a public tip t_i.
4. Buyer 0 observes all the other tips t_i and can offer the proposer of the block a take-it-or-leave-it-offer of a subset $S \subseteq \{1, \ldots, n\}$ of bidders and a bribe p to exclude that subset's bids. Bidder 0 also submits his own bid b_0.
5. The proposer accepts or rejects bidder 0's bribe and constructs the block accordingly, either including $N \setminus S$ if he accepts or N otherwise.
6. The auction is computed based on the bids included in the block.

In the next section, we consider the case of auctions over multiple sequential blocks with independent proposers and the case of simultaneous proposers. Those games are variants of the game above. We describe them in-line.

Formally, pure strategies in this game are:

[1] We abstract away from issues such as block size constraints/ congestion.
[2] This can be practically achieved by submitting the hash of the bid and revealing it later.

- For players $i \in \{1, \ldots, n\}$: A tuple of bidding and tipping strategies $\beta_i : [0,1] \to \mathbb{R}_+$, $\tau_i : [0,1] \to \mathbb{R}_+$ for players $i \in \{1, \ldots n\}$, that is, player i with value v_i bids $\beta_i(v_i)$ and tips $\tau_i(v_i)$.
- For player 0: an offer to the proposer $\theta_0 : \mathbb{R}^n_+ \times [0,1] \to 2^N \times \mathbb{R}_+$, and a bid function $\beta_0 : \mathbb{R}^n_+ \times [0,1] \to \mathbb{R}_+$, i.e. as a function of tips $\mathbf{t} = (\tau_1(v_1), \ldots, \tau_n(v_n))$ and his own value v_0, an offer $\theta_0(\mathbf{t}, v_0)$ and a bid $\beta_0(\mathbf{t}, v_0)$.
- For the proposer, given the tips $\mathbf{t}$ and an offer from player 0, $\theta_0(\mathbf{t}, v_0)$, a choice of which bids to include.

Since our game is an extensive-form game of incomplete information, our solution concept is the Perfect Bayes-Nash Equilibrium (PBE). This requires strategies to be mutual best-responses, as is standard in most equilibria. Additionally, for each player, it requires the player to have beliefs about unknowns at every information set (on-, and off-, path) at which they are called upon to play such that their strategy maximizes their expected utility given the beliefs and others' strategies. Beliefs are correct on path (i.e., derived from the prior, and Bayesian updating given agents' strategies), and unrestricted off path.

Note that the proposer has multiple potential indifferences: e.g., should they include a bid with 0 tip? Should they censor a set of bids if bidder 0's offered bribe exactly equals the total tip from that set? We assume that given tips $\mathbf{t}$ from bidders $\{1, \ldots, n\}$ and an offer from bidder 0 to censor subset S for a bribe of p, the proposer includes the bids of $N - S$ if and only if $p \geq \sum_{i \in S} t_i$, and includes bids from all N otherwise (i.e., we break proposer indifferences in favor of bidder 0 so that best responses are well defined).

There are multiple PBEs of the game, driven in part by the fact that there are multiple equilibria in a second price auction (for instance, there is an equilibrium in the second price auction for one player to bid a high value and all others to bid 0). However, most of these equilibria are in weakly dominated strategies. We therefore focus on the following class of equilibria:

1. Bidders $\{1, \ldots n\}$ submit a truthful bid, i.e. $\beta_i(v_i) = v_i$. Note that this is a weakly dominant strategy for them. In addition, these bidders use a symmetric tipping function τ, that is, $\tau_i(\cdot) = \tau(\cdot)$.
2. Bidder 0 bids equal to his value if he believes, given the tips of $\{1, \ldots n\}$, that there is a nonzero probability that he could win the auction, otherwise he bids 0 or does not bid.

In what follows, we simply refer to a PBE that satisfies this refinement as an *equilibrium* of the game (with no qualifier). We reiterate that there are multiple PBEs of the original game; we are simply restricting attention to these "reasonable" equilibria for tractability.

5 Results

Our results are easiest for the case with 2 bidders. We present this as an illustration before considering the general case.

5.1 Two Bidder Case

Suppose there are only two bidders, one "honest" bidder 1 with value drawn according to distribution F, and one "colluding" bidder who has the opportunity to collude with the proposer, bidder 0, with value drawn independently from distribution F_0. We assume that F_0 satisfies a regularity condition, that is, that $F_0(t)/f_0(t)$ is non-decreasing in t.

The equilibrium in this case is easy to describe:

▶ **Proposition 5.** *The following constitutes an equilibrium of the game with 2 bidders, i.e. $N = \{0, 1\}$, when the seller announces a second-price auction with a reserve price $r = 0$:*

- *Bidder 1 submits a truthful bid, and his tipping strategy as function of his value v_1 is given by $t_1(v_1)$ solves $(v_1 - t) - \frac{F_0(t)}{f_0(t)} = 0$.*
- *Bidder 0's strategy as function of the observed tip t_1 and his value v_0 is given by*

$$\sigma_0(t_1, v_0) = \begin{cases} \textit{bribe} & t_1 \le v_0, \\ \textit{don't bribe} & t_1 > v_0. \end{cases}$$

  where bribe is shorthand for paying t_1 to the proposer in exchange for omitting bidder 1's transaction. Further bidder 0 submits a nonzero bid in the auction if and only if he bribes the proposer.
- *The proposer accepts bidder 0's bribe whenever it is offered and omits bidder 1's bid, otherwise the proposer includes both bids.*

Before providing a proof of this result, the following corollary summarizes the outcome that results in this equilibrium when $F = F_0 = \text{Uniform}[0, 1]$. For comparison, recall that in the (standard) second price auction when both buyers have values drawn i.i.d. from Uniform$[0, 1]$, the expected revenue is $1/3$ and each bidder has an *ex ante* expected surplus of $1/6$.

▶ **Corollary 6.** *Bidder 0 wins the object with probability $\frac{3}{4}$, and has an expected surplus of $\frac{13}{48}$, while bidder 1 wins the auction with probability $\frac{1}{4}$ and has an expected surplus of $\frac{1}{12}$. Revenue for the seller in this auction is 0, and the expected tip revenue for the proposer is $\frac{1}{4}$.*

In short, the proposer collects all the revenue from this auction, while the seller collects none. Bidder 0 is substantially advantaged by his ability to see bidder 1's tip and then decide whether to bribe the proposer or not (wins the auction with higher probability, collects more of the surplus). Our results for $n > 1$ are similarly stark except for the fact that the auctioneer collects some positive revenue when $n > 1$, although this revenue rapidly decreases as n increases.

Proof. The proof of the proposition is straightforward so we describe it briefly in line. First to see that bidder 1 should bid his value (our refinement restricts attention to these) note that bidder 1 has two actions, he privately submits a bid b_1 and publicly submits a tip t. Since bids only matter after inclusion has been decided, which is also after tips have been paid, tips are a sunk cost and what remains is simply a second price auction, in which truthful bidding is a weakly dominant strategy. Therefore it is (part of) an equilibrium for bidder 1 to bid his value.

Notice that bidder 1 does not benefit from submitting a tip $t > v_1$ since even if he wins, he will end up paying more in tips (in addition to possible fees from the auction) than he values the item. Knowing this, it is always weakly better for player 0 to pay t to omit player 1's bid when $v_0 \ge t$. Thus player 0 bribes the proposer if and only if $v_0 \ge t$. In that case player 1's expected utility as a function of his tip is:

$$\mathbb{E}[U_1(v_1, t)] = F_0(t)\,(v_1 - t)\,.$$

Taking the derivative with respect to t, and setting it equal to 0, we get, as desired, that

$$t_1(v_1) \text{ solves } (v_1 - t) - \frac{F_0(t)}{f_0(t)} = 0.$$

Now that we have found a candidate equilibrium, we have some more work to do to verify that it is in fact a PBE. Formally, bidder 1's beliefs at his only information set are that $v_0 \sim \text{Uniform}[0,1]$. By our regularity condition, $t_1(\cdot)$ is strictly increasing. Bidder 0's beliefs, conditional on bidder 1's tip being t_1 are given by

$$v_1 = \begin{cases} t_1^{-1}(t_1) & t_1 \le t_1(1), \\ 1 & \text{otherwise.} \end{cases}$$

Notice that the case of $t_1 > t_1(1)$ is off the equilibrium path. Finally note that Bidder 0's strategy to bribe whenever his value exceeds bidder 1's tip, and to bid only if he is willing to bribe, constitutes a best response. This concludes the proof. ◀

Notice that even though bids are completely private in this model, because of the transaction inclusion micro-structure, bids are effectively revealed by the tips attached to them. This calls into question whether we can conduct sealed bid auctions of any type on chain.

We can also describe the equilibrium for the case where the seller chooses an auction with a reserve price $r > 0$. For brevity we describe this informally:

$$t_1(v_1) \text{ solves } (v_1 - r - t)f_0(r+t) - F_0(r+t) = 0 \text{ if solution exists,}$$
$$t_1(v_1) = 0 \text{ otherwise.}$$

Our regularity condition implies that there exists $\underline{v} = r + \frac{F_0(r)}{f_0(r)} > r$ such that $t_1(v_1) = 0$ for $v \le \underline{v}$ and strictly increasing for $v > \underline{v}$. Bidder 0's strategy is to bribe and submit a bid only if his value $v_0 > t_1 + r$ where t_1 is the observed tip.

▶ **Proposition 7.** *The following constitutes an equilibrium of the game with 2 bidders, i.e., $N = \{0,1\}$, when the seller announces a second-price auction with a reserve price $r > 0$:*

- *Bidder 1 submits a truthful bid, and his tipping strategy as function of his value v_1 is given by $t_1(v_1) = v_1/2 - r$ whenever $v_1 > 2r$ and 0 otherwise.*
- *Bidder 0's strategy as function of the observed tip t_1 and his value v_0 is given by*

$$\sigma_0(t_1, v_0) = \begin{cases} \textit{bribe} & t_1 + r \le v_0 \\ \textit{don't bribe} & \textit{o.w.} \end{cases}$$

  where bribe is shorthand for paying t_1 to the proposer in exchange for omitting bidder 1's transaction. Further bidder 0 submits a non-zero bid in the auction if and only if he bribes the proposer.
- *The proposer accepts bidder 0's bribe whenever it is offered and omits bidder 1's bid, otherwise the proposer includes both bids.*

Note that while the seller does receive positive revenue in this case, they do not realize any "benefit" from running an auction relative to posting a "buy it now" price of r. We formalize this in the following corollary:

▶ **Corollary 8.** *Suppose buyer values are i.i.d. Uniform$[0,1]$. Assuming $r \le \frac{1}{2}$, Bidder 0 wins the object with ex-ante probability $(1-r)(1-(\frac{1}{2}-r)^2)$, while bidder 1 wins the object with ex-ante probability $(1-2r)(\frac{r}{2}+\frac{1}{4})$. The expected revenue of the seller is $r(1-r^2)$, which is the same as the revenue for posting a "buy it now" price of r. The proposer makes an expected revenue of $\frac{1}{4}(1-2r)^2$.*

Again bidder 0 has a strong advantage in this auction. Tips are no longer perfectly revealing, since a non-empty interval of bidder values tip 0, but remain weakly monotone in bid and perfectly revealing when strictly positive.

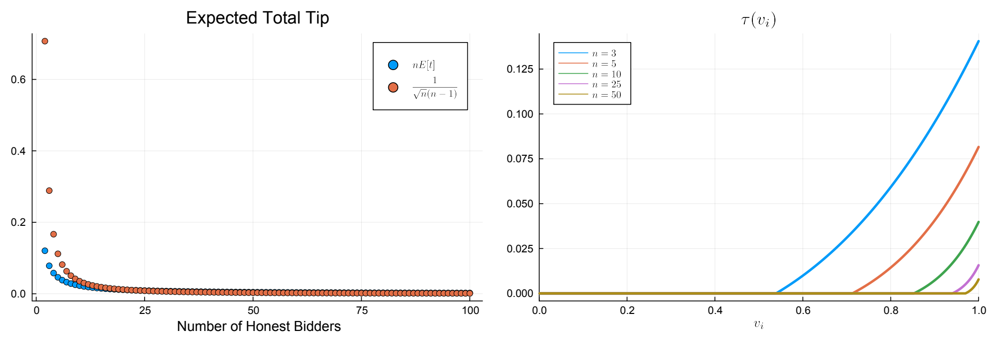


Figure 1 Expected Total Tips and Tipping functions for $F = F_0 \sim \text{Uniform}[0, 1]$.

5.2 Three or more bidders

We now turn to the case where $n \geq 2$, i.e., $N = \{0, 1, \dots, n\}$.

At first the problem of finding an equilibrium may appear intractable since bidder 0's best-response problem is itself complicated: there are 2^n possible subsets of $\{1, \dots, n\}$ and the problem of finding the best subset to buy out is therefore non-trivial. The tipping function of bidders $\{1, \dots, n\}$ needs to be a best response to this (accounting for how changing their tip changes the probability that they are censored given a distribution of other tips). Nevertheless, there is an easy to describe equilibrium. To construct it, the following lemma is useful:

▶ **Lemma 9.** *Suppose that the tipping strategy of buyers $1 \dots n$ is such that $t(v) \leq v/n$. Then we have that the best response for bidder 0, as a function of his own value v_0 and observed vector of tips $\mathbf{t} = (t_1 \dots t_n)$ can be described as:*

$$\sigma_0(v_0, t_1, \dots, t_n) = \begin{cases} \textit{bribe}, & \sum_{i=1}^n t_i \leq v_0 \\ \textit{don't bribe}, & \sum_{i=1}^n t_i > v_0 \end{cases}$$

where bribe is shorthand for paying the proposer $\sum_i t_i$ in exchange for omitting all of bidder 1 through n's transactions.

Proof. To see this note that:

$$\sum_{i=1}^n t(\theta_i) \leq \sum_{i=1}^n \frac{\theta_i}{n} \leq \sum_{i=1}^n \frac{\max(\theta_1, \theta_2, \dots, \theta_n)}{n} = \max(\theta_1, \theta_2, \dots, \theta_n).$$

and therefore bribing the proposer and buying out all the bids (and therefore winning the object for free in the auction) is more profitable than buying out any subset of the bids and possibly losing the auction or having to pay more than the bribes for that subset would have cost. ◀

This lemma is useful because, in equilibrium, the tipping strategy of bidders 1 through n will satisfy this property. Bidder 0's strategy is therefore straightforward (analogous to the case $N = \{0, 1\}$). We are now in a position to describe the equilibrium in this game.[3]

[3] Proofs of this and subsequent propositions are omitted for brevity due to the page limits. They are available from the full version of the paper available on ArXiv, linked on the front page.

▶ **Proposition 10.** *The following constitutes an equilibrium of the game with $n+1$ bidders, $N = \{0, 1, \ldots, n\}$, and buyer values drawn i.i.d. from Uniform$[0,1]$ when the seller announces a second-price auction with a reserve price $r = 0$:*

- *Bidders 1 through n submit truthful bids, and their tipping strategy as function of their value v_i is given by:*

$$t(v) = \begin{cases} 0 & v < \underline{v}, \\ \frac{1}{2n}\left(v^n - \underline{v}^n\right) & o.w. \end{cases} \tag{4}$$

where $\underline{v}$ solves

$$(n+1)\frac{\underline{v}^n}{n(n-1)} - \frac{\underline{v}^{n+1}}{(n+1)} - \frac{1}{n(n+1)} = 0. \tag{5}$$

- *Bidder 0's strategy as function of the observed tips $t_1, \ldots, t_n$ and their value v_0 is given by*

$$\sigma_0(v_0, t_1, \ldots, t_n) = \begin{cases} \textit{bribe}, & \sum_{i=1}^n t_i \leq v_0 \\ \textit{don't bribe}, & \sum_{i=1}^n t_i > v_0 \end{cases}$$

where bribe is shorthand for paying $\sum_i t_i$ to the proposer in exchange for omitting bidder 1 through n's transactions. Further bidder 0 submits a truthful nonzero bid in the auction if and only if he bribes the proposer.

- *The proposer accepts bidder 0's bribe whenever it is offered and omits the other bids, otherwise the proposer includes all bids.*

It is easy to see that (5) has exactly 1 root in $[0,1]$ by observing that the left hand side is increasing in v on $[0,1]$, and evaluates to a negative number at $v = 0$ and a positive number for $\theta = 1$. Unfortunately, we cannot analytically derive these roots for arbitrary n since polynomials of order ≥ 5 do not have explicit roots (and even for $n = 2, 3$ these are not particularly nice); however, we can use a zero finding algorithm to compute these numerically. The results for the uniform case are presented in Figure 1.

Analytically, we can bound how this root varies with n. Note that the expected total tip is $n\mathbb{E}[t(v)]$ and substituting in (4) and simplifying via (5), we have that the expected total tip $= \frac{\underline{v}^n}{n-1}$. The following proposition describes the asymptotic behavior of the expected total tip:

▶ **Proposition 11.** *Let $\underline{v}(n)$ describe the solution to (5) as a function of n. There exists $\underline{n}$ large enough such that for $n > \underline{n}$, we have $\frac{1}{n} \leq \underline{v}(n)^n \leq \frac{1}{\sqrt{n}}$.*

Proposition 11 is particularly useful when you consider the fact that for large n, by the law of large numbers, the total tip concentrates around the expected tip with high probability (buyer values are i.i.d. bounded random variables). Furthermore, by Proposition 11, this is decreasing at a rate at least $1/n\sqrt{n}$: as n grows, and individual bidders are willing to tip less. To see why tipping is only profitable when it leads to the bid not being censored *and* winning the auction, but increasing the tip increases the probability that all bids are not censored. In short, tips have public goods type properties. Indeed, the rate of tipping shrinks fast enough so that the total tip is also decreasing. Therefore bidder 0 wins the auction with increasing probability in n, asymptotically tending to 1. Note that the seller only receives revenue when there is more than one bidder in the auction (or more generally, in the auction with a reserve price r, makes revenue larger than the reserve price) – and this happens with vanishing probability as n grows large.

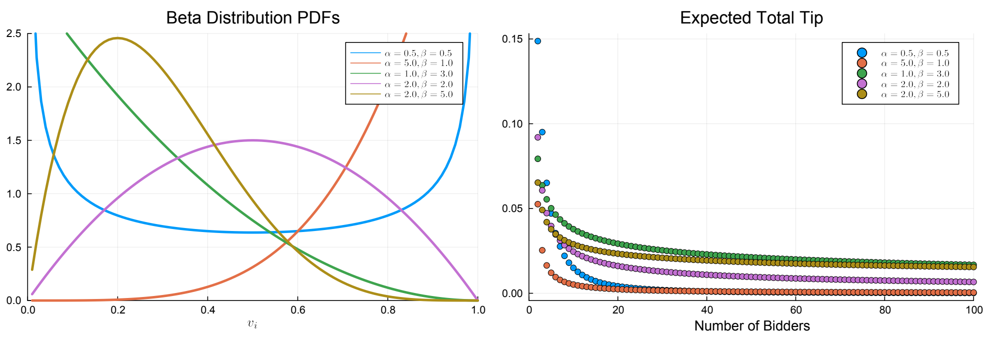


Figure 2 Expected Total Tips for $F_0 \sim \text{Uniform}[0,1]$, $F \sim \text{Beta}(\alpha, \beta)$.

▶ **Proposition 12.** *As n grows large, the expected revenue of the auction with reserve price r reduces asymptotically to the expected revenue of a published price of r.*

5.3 General Distributions

It is straightforward to generalize our results to the case where bidders $\{1, \dots, n\}$ are distributed according to some general distribution with density f and CDF F on $[0, 1]$.

▶ **Assumption 13.** *We assume throughout that F satisfies $\int_0^v F^{n-1}(\theta)d\theta \leq \frac{v}{n}$ for all $v \in [0, 1]$.*

Formally, we have the following proposition:

▶ **Proposition 14.** *The following constitutes an equilibrium of the game with $n+1$ bidders, i.e. $N = \{0, 1, \dots, n\}$, when the seller announces a second-price auction with a reserve price $r = 0$, bidder 0 has a value Uniform$[0, 1]$ and bidders 1 through n have values distributed i.i.d. according to a distribution with density f and CDF F satisfying Assumption 13:*

- *Bidders 1 through n submit truthful bids, and their tipping strategy as function of their value v_i is given by:*

$$t(v) = \begin{cases} 0 & v < \underline{v}, \\ \frac{1}{2}\int_{\underline{v}}^{v} F^{n-1}(\theta)d\theta & o.w. \end{cases} \tag{6}$$

where $\underline{v}$ solves

$$\int_0^1 F^{n-1}(\theta)d\theta - \int_{\underline{v}}^1 F^n(\theta)d\theta = \frac{n+1}{n-1}\int_0^{\underline{v}} F^{n-1}(\theta)d\theta. \tag{7}$$

- *Bidder 0's strategy as function of the observed tips $t_1, \dots, t_n$ and their value v_0 is given by*

$$\sigma_0(v_0, t_1, \dots, t_n) = \begin{cases} bribe, & \sum_{i=1}^n t_i \leq v_0 \\ don't\ bribe, & \sum_{i=1}^n t_i > v_0 \end{cases}$$

where bribe is shorthand for paying $\sum_i t_i$ to the proposer in exchange for omitting bidder 1 through n's transactions. Further bidder 0 submits a true nonzero bid in the auction if and only if he bribes the proposer.
- *The proposer accepts bidder 0's bribe whenever it is offered and omits the other bids, otherwise the proposer includes all bids.*

This proposition allows us to numerically solve for tipping behavior in this auction. Using the flexible Beta distribution for a range of parameters, we compute $n\mathbb{E}[t]$ as a function of n in Figure 2.

Analytical results for the case where bidder 0's value is distributed non-uniformly appear out of reach. To see why – for bidders 1 through n, part of the payoff of increasing their tip is increasing the probability that bidder 0 chooses not to buy them out. When bidder 0's value is distributed uniformly, the increase in probability is constant and independent of others' tips (which from the perspective of the bidder is a random variable). This simplifies the optimality condition and makes it analytically tractable. Nevertheless, the intuition above suggests that our qualitative results (seller expected revenue drops to close to the posted price, tips do not offer much "protection" due to the public goods nature of tips suggests, bidder 0 has a strong advantage in the auction) carry over to this case as well.

6 Restoring Censorship Resistance

We now discuss possible design choices to provide additional censorship resistance, so that an auction can be run on chain with the desired results.

6.1 Auction over Multiple Blocks

We now investigate whether running the auction over multiple blocks restores the desired behavior. Formally, recall that using multiple blocks as the underlying public bulletin board had a higher censorship resistance (Example 3) than in the case of a single block (Example 2).

Formally we consider the following dynamic game corresponding to an auction being run over m blocks. We assume that each of these blocks is produced by an independent proposer.[4]

1. Period 0: Bidders learn their values v_i. Bidders $1, \ldots, n$ each submit simultaneously a private bid b_i and a public tip t_i.
2. Period j for j in 1 to k: Bidder 0 observes which bids from $1, \ldots n$ have not been included in a block in periods 1 to $j-1$. They offer Proposer j a take-it-or-leave-it-offer of a subset S_j of the unincluded bids and a payment p_j to exclude that subset. The proposer j observes the tips and the offer from Bidder 0 and decides which bids if any to include.
3. Period $m+1$: The seller's auction is run on blocks produced in periods 1 to m.

If a transaction is included in period j, it is removed from the set of bids in the mempool, its tip is attributed to proposer j where j is the block it was included in, it is included in the auction, and it cannot be included in subsequent blocks.

Note that the game we describe is a natural extension of the previous game to the case of an auction over $m > 1$ blocks. In particular, it reduces to the original game for the case of $m = 1$. We consider the same refinement as before, which applies to this game in a similar fashion.

Note that there are two possible sources of additional security in this auction. The first is mechanical: in order to censor a transaction, intuitively, bidder 0 has to bribe m proposers, which is more expensive for a given tip. The second is that the marginal returns to a tip have increased (increasing a tip by q increases the cost to censor for bidder 0 by mq, and decreases the probability they can afford it correspondingly).

[4] In practice, an majority of blocks on major blockchains is produced by one of a small oligopoly of proposers. We discuss the implications of this in the sequel.

In our results, we show that the latter effect is null. In particular, we show that for $m < n$, the tipping behavior of bidders $1, \ldots n$ stays unaffected.

Formally, we have the following result:

▶ **Proposition 15.** *Suppose buyers* 1 *to* n *have values drawn i.i.d.* $U[0,1]$ *and buyer* 0 *has value drawn i.i.d.* $U[0,\kappa]$ *for* $\kappa > m$*. The following constitutes an equilibrium of the game with* $n+1$ *bidders, i.e.* $N = \{0, 1, \ldots, n\}$*, when the seller announces a second-price auction with a reserve price* $r = 0$ *to be run over* m *blocks for* $m < n$*: Bidders 1 through* n *and the proposers have the same strategy as in Proposition 10.*

Bidder 0*'s strategy as function of the observed tips* $t_1, \ldots, t_n$ *and his value* v_0 *is given by*

$$\sigma_0(v_0, t_1, \ldots, t_n) = \begin{cases} \textit{bribe}, & m \sum_{i=1}^{n} t_i \leq v_0, \\ \textit{don't bribe}, & \textit{otherwise}, \end{cases}$$

Before we proceed, we comment on the assumption that bidder 0's value is distributed $U[0,\kappa]$. Suppose bidder 0 is distributed $U[0,1]$, but bidders 1 to n tip as in Proposition 10. Note that with positive probability the total tip could exceed 1 when $m > 1$. Therefore, given bidders 2 to n follow the tipping strategy of Proposition 10, bidder 1 will have incentives to shade their tip relative to $t(\cdot)$. After all, the marginal value of tips depends on how much they increase the probability of not being censored. From the Proof of 10, in the case of $m = 1$, increasing one's tip on the margin always increases the probability that the bids are not censored, because the total tip is strictly smaller than 1 with probability 1 on path. Intuitively, therefore if bidder 0 is distributed $U[0,1]$, and the auction is conducted over $m > 1$ blocks, the equilibrium tipping strategy for bidders 1 to n is weakly lower than the corresponding tipping strategy for $m = 1$ (Proposition 10). This can be shown numerically, but is out of reach analytically.

Note that we had already shown that expected total tip of n bidders was smaller than $1/n^{3/2}$. Therefore, the probability that bidder 0 does not censor the remaining bids collapses to 0 as n grows large, as long as m grows sublinearly with n. To see that we had already shown that expected total tip of n bidders was smaller than $1/n^{3/2}$. Therefore m times this for $m < n$ still grows smaller than $1/\sqrt{n}$.

Put differently, guaranteeing the auction outcome is "as desired" requires $m > n$. This comes with its own costs: for example, the auction would have to remain open for a relatively long time which may be undesirable, particularly for financial applications.

6.2 Multiple Concurrent Block Proposers

Depending on the number of bidders and the time constraints inherent to the specific auction application, it may not be feasible to hold the auction for long enough to achieve the desired censorship resistance level.

A different solution we now consider would be to allow more than one proposer within a single slot. Formally, we now consider k concurrent block proposers (by analogy to our previous section where we considered k sequential block producers). The seller announces an auction which will execute within the single slot, i.e. the bids included on at least one of the k concurrent produced blocks will be included in the auction.

In view of the concurrency, we allow bidders to submit conditional tips, which depend on the number of proposers who include the transaction. For simplicity, we consider a *twin tip*, i.e., each bidder submits a conditional tip of the form (t, T), where T is paid if only a single proposer includes bidder 1's transaction and t is paid if more than one proposer includes the transaction.

▶ **Observation 16.** *With k concurrent proposers and conditional tipping, the censorship resistance of a conditional tip (t, T) is straightforwardly verified as:*

$$\varphi(t, T) = kT. \tag{8}$$

It is important to note that the conditional tip disentangles the cost of inclusion (for the transacting party) from the cost of censoring, i.e. if $T \gg t$, then the censorship resistance is kT which is much larger than the cost of inclusion, kt.

After the honest bidder observes v_1 and submits his private bid b_1 and public tip (t, T), the bribing bidder submits a bribe to each proposer. Formally, we first consider the following game:

1. Seller announces second-price sealed-bid auction with reserve price r to be conducted over a single slot.
2. Buyers learn their values $v_i \sim F$.
3. Buyers $1, \ldots, n$ each submit simultaneously a private bid b_i and a public tip t_i, T_i.
4. Buyer 0 observes all the other tips t_i, T_i and simultaneously offers each proposer a take-it-or-leave-it-offer of a subset $S \subseteq \{1, \ldots, n\}$ of bidders and a payment p to exclude that subset's bids. Bidder 0 also submits his own bid b_0.
5. Each Proposer simultaneously accepts or rejects bidder 0's offer and constructs the block accordingly i.e., either containing bids of set $N \setminus S$ or N.
6. The auction is computed based on the union of the bids included in all blocks, tips are paid based on the inclusion behavior.

As before we focus our attention on equilibria where Bidders $\{1, \ldots, n\}$ bid truthfully in the auction. Note that each proposer, in choosing whether to censor a transaction, needs to reason about the behavior of other proposers since that potentially affects their tip if they include the transaction.

▶ **Proposition 17.** *Consider the Multiple Concurrent Block Proposer game, with m proposers and $n = 1$ honest bidder, i.e., $N = \{0, 1\}$, with bidder i's value drawn from a distribution with CDF F_i and PDF f_i.*

This game has an equilibrium where the outcome of the auction is the same as a standard second price auction without a censorship step and where the expected tip by each bidder to each proposer is $\underline{t}$.

In particular, bidders 1's tipping strategy in equilibrium is given by:

$$t_1(v_1) = 0, \qquad\qquad T_1(v_1) = 1 \tag{9}$$

Bidder 0's offer strategy to the proposer based on their own value v_0 and the observed tips (t, T) is

$$z_0(t, T, v_0) = \begin{cases} 0 & C(v_0) < mT \\ T & C(v_0) \geq mT. \end{cases} \tag{10}$$

Here $C(v_0)$ is buyer 0's net value to censoring bidder 1's bid (i.e., the difference their profit from censoring the competing bid and winning in the auction for free (v_0); and their expected surplus from competing with bidder 1 in the auction). Finally, the proposer's strategies are to censor transactions with the following probabilities:

$$p(z, t, T) = \begin{cases} 0 & z < t \\ \left(\frac{z-t}{T-t}\right)^{\frac{1}{m-1}} & t \leq z < T \\ 1 & z \geq T \end{cases} \tag{11}$$

The above proposition may admittedly appear a little dense. We provide the following corollary regarding (on-path) equilibrium behavior.

▶ **Corollary 18.** *Consider the equilibrium proposed in Proposition 17 for any $m \geq 2$. On path, bidder 0 does not bribe the proposers and instead competes in the second-price auction. All bidders pay 0 in tips on path. Equilibrium tips do not reveal bids.*

Further, a careful study of the proof of Proposition 17 shows that the Corollary continues to hold even when $n > 1$. Therefore even 2 concurrent block proposers restore the "desired" outcome relative to a single block proposer system. This is partly driven by concurrency of block proposers which removes the "monopoly" that they have over transaction inclusion, and partly by the conditional tip. The conditional tip allows bidder 1 to get security via a high-tip offer conditional on inclusion by only a single proposer. This high tip offer makes it very expensive for bidder 0 to attempt censor bidder 1's bid since they would need to pay m times the high tip to censor the bid by all m producers. Therefore, no bribe is offered. Further, this tip never needs to be paid, since both proposers find it weakly dominant to include the bid and pick up the low tip.[5] As an aside, note that this also restores equilibrium bid privacy, since tips no longer reveal bids.

Finally, we should note that the conditional tipping logic we identify could also be applied to an auction over multiple blocks, achieving more censorship resistance than previously.

7 Discussion

Our results suggest that single proposer blockchains are not ideal for holding time sensitive auctions when the number of potential bidders is large. In our results, collusion arrangements are extremely profitable for the colluding bidder but only marginally profitable for the proposer. However, this is because we restrict the model to have one potential colluding bidder who can bribe the proposer. In reality, there are many possible colluding bidders, and only one proposer in each slot, the proposer could end up charging for the right to collude and extract a significant portion of the value that the colluding bidder gains from the arrangement. In fact, the predominant block building system, MEV-boost, can be thought of as a direct channel through which the proposer can sell the right to censor transactions to the highest bidder. This suggests that one driver of MEV is the proposer's right to determine inclusion. Previous work has focused on a different source: the proposer's right to order transactions within a block. From the position that proposer ordering power is the source, order agnostic mechanisms should solve MEV. But if censorship power is the source, these order agnostic mechanisms, including the second price auction we study here, could be just as susceptible to value extraction.

Another proposed source for MEV is the public nature of transactions in the mempool. The argument is, transactions in the mempool are sitting ducks, waiting to be front-ran. It follows that, if transactions are encrypted while in the mempool, they will be less susceptible to MEV. But our results demonstrate that even when the body of transactions are encrypted, public tips may reveal substantial information them.

[5] Note that $t = 0$ supports alternative asymmetric equilibria in the inclusion subgame where only a single proposer includes the transaction. In the broader game, these would correspond to the equilibria in the single proposer cases where T substitutes for the old single dimensional tip. However, when t is bounded away from 0, these equilibria disappear, since it is now strictly dominant for the proposer to include the bid in the inclusion subgame.

As the previous paragraph suggests, cryptographic solutions do not immediately solve the censorship problem unless they also appropriately modify consensus to bind the proposer (*cf* our multiple concurrent proposer suggestion): for example consider an MPC based approach. One way to implement it would be to use an integer comparison MPC protocol to compare the private tips and choose the top k. Then, if the proposer chose not to include those top k, they could be penalized, or the block could be considered invalid by consensus. Note that this requires changing consensus as well as the mempool, and it requires multiple validators to participate in MPC, each of whom can add transactions to the process. In that way it is similar to the multiple concurrent block proposer proposer scheme we describe. However, in this scheme, the proposer also retains the option to accept a bribe rather than use the MPC protocol, i.e., they may accept a bribe that exceeds the expected value of the top k tips.

Implemented purely as a private mempool solution without changing consensus, the main impediment is enforcing the proposer's commitment to this block inclusion strategy: i.e., MPC allows us to compute the k transactions with the highest tips in a private fashion, but does not bind the proposer to include these. The proposer remains free to censor any subset of these transactions if it is profitable for them to do so.

7.1 Multiple Concurrent Proposers in the Real world

There are projects trying to implement multiple concurrent proposers by appropriately modifying the Tendermint protocol.[6] Representatives from at least one major chain, have also mentioned concurrency as a goal going forward, in order to scale throughput and decrease latency from the user to the nearest proposer within a given slot.[7] The co-founder, and CEO of Solana even mentioned MEV resistance as a motivation for the desirability of multiple concurrent block proposers.[8] Our results suggest that proposer rents arising from the proposer's temporary monopoly on inclusion shrink sharply under multiple concurrent proposers using our conditional tip logic. To see the order of magnitude, note that total payments by MEV-boost to proposers on Ethereum totaled \$400 million since Sept 2022 (the merge), see `https://dune.com/arcana/mev-boost`). This is quite large in comparison to the total value created by MEV and contributes to high transaction fees for users. That said multiple proposer protocols are not a panacea. Beyond the engineering difficulties themselves, we see a few other impediments. Firstly, ordering – with a single proposer, a canonical ordering of transactions within the block is simply the order that the leader put them in. Once we add multiple proposers, the ordering becomes less clear. Secondly, multiple proposers may be more susceptible to Distributed Denial of Service (DDOS) attacks: allowing every validator to submit transactions opens the network up to a DDOS attack where adversaries include many transactions to slow down the chain. Finally there are issues of redundancy and state bloat: the multiple proposer approach potentially leads to considerable redundancy within the slot (the same transaction may be included by many different proposers).

[6] See, e.g., `https://blog.duality.xyz/introducing-multiplicity/`.
[7] See, e.g., `https://blog.chain.link/execution-and-parallelism-for-dag-based-bft-consensus/`.
[8] See `https://twitter.com/aeyakovenko/status/1584676110948012032`.

7.2 Future Directions

From a theoretical perspective, this work leaves open questions of how censorship in on chain auctions might effect the equilibria of auctions with different assumptions about bidder valuations. For example a natural extension to our results would be to consider honest bidders with interdependent or common values. This may be a better model for on chain order flow auctions and collateral liquidation auctions.

Our results are based on the assumption that a single bidder has been selected as the colluding bidder in advance, but a better assumption would be to have the right to collude be auctioned off after bids have been submitted. If the right to collude is auctioned off before the bids are submitted, then our results would still hold, since our result would simply be a subgame of the larger game being considered, and the result would be that proposers end up with a larger share of their monopoly rents; however, when the right to collude is auctioned off after transactions have been submitted, and therefore after bidders discover their types, the players who are willing to pay to collude are more often those who value the item more. This could warp the equilibrium slightly.

Outside of mechanism design, this work provides a strong theoretical justification for investigating multiple concurrent block proposer based consensus frameworks as a tool for MEV mitigation. Specifically, we identify conditional tipping as a powerful tool to combat censorship in situations where there are more than one block proposer.

Indeed, strong censorship-resistance is beneficial to other systemically important smart contracts. Oracle feeds require timely inclusion guarantees to be useful, and censorship in order to manipulate on-chain financial markets is a serious concern. Hedging contracts such as options need to be censorship resistant so that they can be exercised, and conversely, the underwriter has an incentive to censor. Similarly, optimistic rollups rely on censorship resistance for their security, an attacker who wishes to manipulate an optimistic rollup only needs to censor the fraud proofs for a period of time. To combat this, optimistic rollups currently leave a long window for fraud proof submission before finalizing (e.g. 7 days). Stronger censorship resistance could allow them to achieve finality faster with the same security.

Another advantage is that proposer rents arising from the proposer's temporary monopoly on inclusion shrink sharply under multiple concurrent proposers using our conditional tip logic. To see the order of magnitude, note that total payments by MEV-boost to proposers on Ethereum totaled \$445 million since Sept 2022 (the merge), see `https://dune.com/ChainsightAnalytics/mev-after-ethereum-merge`). This is arguably quite large in comparison to the total value created by MEV and contributes to high transaction fees for users.

Another potential tool for combating censorship on-chain, that we have not discussed, is a data availability layer. Instead of being submitted to a blockchain directly, bids could be submitted to a data availability layer, nodes could then compute the results of the auction based on whichever transactions were included on the data availability layer. This is similar to holding the auction directly on chain except that the nodes tasked with curating a data availability layer do not necessarily need to participate in consensus. The requirements for a data availability layer are weaker than those required for a full blockchain so it may be easier to integrate multiple proposer architecture on data availability layers than on blockchains themselves.

References

1 Mohammad Akbarpour and Shengwu Li. Credible mechanisms. In *EC*, page 371, 2018.

2 Mohammad Akbarpour and Shengwu Li. Credible auctions: A trilemma. *Econometrica*, 88(2):425–467, 2020.

3 Erik-Oliver Blass and Florian Kerschbaum. Strain: A secure auction for blockchains. In *European Symposium on Research in Computer Security*, pages 87–110. Springer, 2018.

4 Eric Budish, Peter Cramton, and John Shim. The high-frequency trading arms race: Frequent batch auctions as a market design response. *The Quarterly Journal of Economics*, 130(4):1547–1621, 2015.

5 Vitalik Buterin. The problem of censorship. `https://blog.ethereum.org/2015/06/06/the-problem-of-censorship`, June 2015.

6 Vitalik Buterin. State of research: increasing censorship resistance of transactions under proposer/builder separation (pbs). `https://notes.ethereum.org/@vbuterin/pbs_censorship_resistance`, June 2021.

7 Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 719–728, 2017.

8 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

9 Edith Elkind and Helger Lipmaa. Interleaving cryptography and mechanism design: The case of online auctions. In *Financial Cryptography: 8th International Conference, FC 2004, Key West, FL, USA, February 9-12, 2004. Revised Papers 8*, pages 117–131. Springer, 2004.

10 Matheus VX Ferreira and David C Parkes. Credible decentralized exchange design via verifiable sequencing rules. *arXiv preprint arXiv:2209.15569*, 2022.

11 Matheus VX Ferreira and S Matthew Weinberg. Credible, truthful, and two-round (optimal) auctions via cryptographic commitments. In *Proceedings of the 21st ACM Conference on Economics and Computation*, pages 683–712, 2020.

12 Flashbots. The future of mev is suave: Flashbots, November 2022. URL: `https://writings.flashbots.net/the-future-of-mev-is-suave/#iii-the-future-of-mev`.

13 Frankie, Dan Robinson, Dave White, and andy8052. Gradual dutch auctions, April 2022. URL: `https://www.paradigm.xyz/2022/04/gda`.

14 Hisham S Galal and Amr M Youssef. Verifiable sealed-bid auction on the ethereum blockchain. In *Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers 22*, pages 265–278. Springer, 2019.

15 Nicholas A. G. Johnson, Theo Diamandis, Alex Evans, Henry de Valence, and Guillermo Angeris. Concave pro-rata games, 2023. `arXiv:2302.02126`.

16 Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *Cryptology ePrint Archive*, 2021.

17 Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Annual International Cryptology Conference*, pages 451–480. Springer, 2020.

18 Kshitij Kulkarni, Theo Diamandis, and Tarun Chitra. Towards a theory of maximal extractable value i: Constant function market makers. *arXiv preprint arXiv:2207.11835*, 2022.

19 Kshitij Kulkarni, Matheus VX Ferreira, and Tarun Chitra. Credibility and incentives in gradual dutch auctions. *arXiv*, 2023.

20 Byoungcheon Lee, Kwangjo Kim, and Joongsoo Ma. Efficient public auction with one-time registration and public verifiability. In *Progress in Cryptology—INDOCRYPT 2001: Second International Conference on Cryptology in India Chennai, India, December 16–20, 2001 Proceedings 2*, pages 162–174. Springer, 2001.

21 Jason Milionis, Dean Hirsch, Andy Arditi, and Pranav Garimidi. A framework for single-item nft auction mechanism design. In *Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security*, pages 31–38, 2022.

22 Barnabe Monnot. Unbundling pbs: Towards protocol-enforced proposer commitments (pepc). `https://ethresear.ch/t/unbundling-pbs-towards-protocol-enforced-proposer-commitments-pepc/13879?u=barnabe`, October 2022.

23 Tim Roughgarden. Transaction fee mechanism design. *CoRR*, abs/2106.01340, 2021. `arXiv:2106.01340`.

24 Koutarou Suzuki and Makoto Yokoo. Secure generalized vickrey auction using homomorphic encryption. In *Financial Cryptography: 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003. Revised Papers 7*, pages 239–249. Springer, 2003.

25 Nick Szabo. Formalizing and securing relationships on public networks. *First monday*, 1997.

26 Anton Wahrstätter, Jens Ernstberger, Aviv Yaish, Liyi Zhou, Kaihua Qin, Taro Tsuchiya, Sebastian Steinhorst, Davor Svetinovic, Nicolas Christin, Mikolaj Barczentewicz, et al. Blockchain censorship. *arXiv preprint arXiv:2305.18545*, 2023.

The Centralizing Effects of Private Order Flow on Proposer-Builder Separation

Tivas Gupta
Special Mechanisms Group, USA

Mallesh M. Pai
Department of Economics, Rice University, Houston, TX, USA
Special Mechanisms Group, USA

Max Resnick
Special Mechanisms Group, USA

Abstract

The current Proposer-Builder Separation (PBS) equilibrium has several builders with different backgrounds winning blocks consistently. This paper considers how that equilibrium will shift when transactions are sold privately via order flow auctions (OFAs) rather than forwarded directly to the public mempool. We discuss a novel model that highlights the augmented value of private order flow for integrated builder searchers. We show that private order flow is complementary to top-of-block opportunities, and therefore integrated builder-searchers are more likely to participate in OFAs and outbid non integrated builders. They will then parlay access to these private transactions into an advantage in the PBS auction, winning blocks more often and extracting higher profits than non-integrated builders. To validate our main assumptions, we construct a novel dataset pairing post-merge PBS outcomes with realized 12-second volatility on a leading CEX (Binance). Our results show that integrated builder-searchers are more likely to win in the PBS auction when realized volatility is high, suggesting that indeed such builders have an advantage in extracting top-of-block opportunities. Our findings suggest that modifying PBS to disentangle the intertwined dynamics between top-of-block extraction and private order flow would pave the way for a fairer and more decentralized Ethereum.

2012 ACM Subject Classification Applied computing → Online auctions

Keywords and phrases Private Order Flow, PBS, OFAs, decentralization

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.20

Related Version *Full Version*: `https://arxiv.org/abs/2305.19150`

1 Introduction

Most Ethereum blocks today are built by specialized *builders* rather than validators. In every slot, builders gather transactions and assemble them into blocks. They then compete against each other in an ascending price (English) auction for the right to have the block they assembled proposed by the proposer. Whichever builder bids the highest wins the Proposer-Builder Separation (PBS) auction, and pays their bid to the proposer.

The right to build a block is valuable for several reasons, most obviously because users pay *tips* for inclusion. Presently these tips make up only a small portion of the total value from building a block. A majority of the value from building a block comes from the builder exploiting *MEV opportunities.* MEV (Maximal Extractable Value) refers to additional value that can be exploited from strategically reordering or including specific transactions.

Current MEV opportunities on Ethereum can be broadly segmented into two categories: *top-of-block* and *block body.* Let us describe each in turn. Top-of-block opportunities are primarily CEX/DEX arbitrage: exploiting price divergences of a token between a centralized exchange (CEX) and some on-chain Decentralized Exchange (DEX) operated by a smart contract, e.g., Uniswap. Intuitively, successfully exploiting such a price divergence requires both priority access to the first few transactions in the block on-chain, and also high quality execution on the centralized exchange. The latter requires high-frequency trading (HFT) strategies and low CEX transaction fees.

Block-body opportunities are typically frontrunning attacks that involve sandwiching user transactions or executing user orders against each other to cut out the liquidity providers. The value of the Block-body is primarily dictated by access to transactions. Historically, most transactions have been forwarded to the public mempool, meaning all block builders have access to the same transactions; however, some builders have access to private order flow which is not available in the public mempool. The availability of private order flow is likely to be further supplemented in the near future by the advent of order flow auctions (OFAs), venues where order flow providers (wallets) sell the exclusive right to execute their users' transactions.

This paper focuses on the complementarity between top-of-block and block body opportunities. In particular, the PBS auction makes no distinction between top-of-block and block body, instead, the right to build the entire block is sold wholesale. This means that an advantage in top-of-block extraction capability can help secure value from the body of the block and vice versa.

This paper makes two contributions. First, we demonstrate empirically that builders operated by high-frequency trading firms are superior at capturing the top of block opportunities. Second, we construct a simple model of proposer-builder separation and demonstrate that, in this model, private order flow is more valuable to vertically integrated builder searchers than non-integrated builders. Our theoretical results therefore imply that private order flow markets are likely to be dominated by these firms.

Let us now describe our analysis and results in a little more detail. The main assumption in our subsequent theoretical analysis is that some bidders are stochastically advantaged at extracting top-of-block opportunities. We validate this assumption empirically. In particular, we construct a unique dataset that combines roughly a month PBS auction outcome data, i.e., which builder won which blocks over the course of a month; paired with detailed price data on a major CEX, namely, Binance. Our empirical strategy posits that the realized 12-second volatility of ETH on Binance is plausibly exogenous. Therefore the realized volatility will generate blocks that have varying top of block value. A block in which the price on Binance is flat over the previous 12 seconds will have almost no top of block value, meaning any advantage that builder searchers have at extracting from the top of the block will be irrelevant. In contrast, if the price shift is large in that period, winning the block becomes should be far more valuable to builders who excel at top-of-block extraction. Our results show that when absolute log price change on Binance in a 12 second period is large, builder-searchers operated by HFTs are far more likely to win. These results are statisitcally significant, invalidating the null hypothesis that all builders are roughly equivalent in their top-of-block extraction capability.

Having demonstrated this, we turn to a theoretical model which explores the centralizing effects of this top-of-block advantage on the equilibrium of the PBS auction. Our model considers a simple abstraction where a block can contain at most two transactions.

In the first stage of our model, the builders gather block body opportunities. We consider two scenarios. In the first, this transaction is sourced from the public mempool, i.e. all builders have (free) access to the block-body transactions. This models relatively well the current state of affairs. The second scenario envisages builders purchasing transactions in an OFA, which models the plausible scenario we are moving towards. In the second stage of our model, the builders combine their block body transactions with their top-of-block transactions to form a block and then compete with each other in an English (i.e., ascending) auction for the right to append their block onto the chain.

Our results show that advantages in top of block extraction capabilities are magnified when private order flow is available, in comparison with the current scenario where block body opportunities are available in the public mempool or otherwise shared with all builders. In particular, a builder with an advantage, be it deterministic or stochastic, at extracting top-of-block opportunities, will win the OFA. With access to the private transactions, it will then win the PBS auction more often, and have higher profits, than it would have in the counterfactual world without OFAs/ private transactions.

Our results suggest a troubling centralizing tendency of PBS when private order flow is available via an Order Flow Auction, a setting we are moving towards. In particular, a small number of integrated builder-searchers who have top-of-block extraction capabilities will dominate both the OFAs and the downstream PBS auction. This contrasts with the popular idea that the OFAs and the PBS auction will squeeze proposer profits between the validators (who earn the PBS auction revenue) and the order flow providers (who earn the OFA revenue). This also contrasts with the original goal of PBS which was to keep block building decentralized.

Our results therefore provide a further impetus for various initiatives to "unbundle" PBS – unbundling PBS in some form is necessary to prevent concentration into a few integrated builder-searchers. Previous work has focused on limiting the power of builders to build blocks by imposing certain constraints on them: see e.g. the recent works of [1], [11]. There have also been studies on the possibility of implementing blockspace futures (see, e.g., [7]), which would effectively partially disintermediate the builder by guaranteeing inclusion for some transactions.

2 Related Literature

Although Proposer Builder Separation has only recently become the dominant method of building blocks on Ethereum, research into PBS dynamics is active with several recent developments. [4] discusses the potential for exclusive order flow to centralize the builder market. [5] surmised that order flow auctions could potentially alleviate the centralizing effects of private order flow by providing a level playing field for all builders to bid in; however, as we discuss later, our results suggest that order-flow auctions may still have a centralizing effect on PBS because builders with advantages in top of block extraction may come to dominate the auction, resulting in equilibria that look similar to those where private order flow is purchased by a single entity. [2] catalogued several competing order-flow auction designs and outlined their respective advantages and disadvantages.

A series of articles have provided visibility into the current status of the MEV supply chain. [3] discussed to what degree relays (which forward PBS bids to the proposer) have lived up to their promises about which types of transactions and MEV strategies are allowed in their blocks. [15] catalogues the current state of the PBS market, noting which builders submit to which relays, and providing insight into the total revenue from the PBS system so far. [13] discusses how proposers who participate in MEV boost could raise their revenue by delaying block proposal, allowing more bids to come in before choosing a winner. [14] uses proprietary data provided by Titan builder to demonstrate that top builders have more order flow than other builders and that this is a large factor contributing to their dominance in the PBS auctions.

Recent developments have increased our understanding of CEX/DEX arbitrage also known as *loss-versus-rebalancing* (LVR) or stale order sniping. [10] proposed the definition for LVR as the loss that the pool incurred relative to a perfect re-balancing portfolio. This quantity can also be thought of as the expected profit of top of block CEX/DEX arbitrage bots. [9] extended this analysis from continuous time with no fees to discrete time with fees, a much more realistic model. A core finding of this paper was the result that LVR grows cubically in blocktime, with smaller blocktimes leading to lower LVR.

3 Background

The easiest way to understand the top-of-block, block-body distinction is to look at the blocks themselves. CEX/DEX arb transactions are easily identifiable since they are large directional trades, typically in the first few slots of the block.

These CEX/DEX arb transactions are usually executed by an MEV bot contract that disproportionately lands transactions in blocks associated with the corresponding builder. For example, block 17195495,[1], contains 182 transactions. The first 37 appear to be CEX/DEX arb transactions from an MEV bot with the address 0xA69b…e78C.[2] These are subjectively large swaps on major pools (Uniswap, Sushiswap etc). For example, the first transaction swaps 4.265 Million USDC for 2168 wETH [3] on the Uniswap v3 0.05% fee pool.[4] The subsequent 36 are also similarly large swaps, each of the order of several hundred wETH.

Note that these CEX/DEX arbitrage transactions are not found on all blocks – for example, the preceding block, 17195494, does not contain such transactions. They typically only appear when there is high volatility in the preceding 12 seconds, and even then, the sizes tend to be much smaller than this selected block in most cases. For example, in the next block 17195496, there is only 1 CEX/DEX arb transaction from the same bot and the volume traded is only 1.2 Million USDC for 600 WETH.[5]

In the block after that, block 17195497, the same bot has a single CEX/DEX arb transaction, swapping 272k USDT for 138 ETH.[6] After this transaction, the rest of the block is filled with block-body opportunities. Transactions at indexes 1–4 and 11–14 are sandwich attacks.[7]

Block 17195497 in particular shows that builders can exploit both top-of-block and block-body opportunities in the same block. This is an important aspect of our model, and drives our results.

[1] See, e.g., https://etherscan.io/block/17195495.
[2] 0xA69babEF1cA67A37Ffaf7a485DfFF3382056e78C
[3] https://eigenphi.io/mev/eigentx/0xca8ec486cb46066b464104c1b91b3e253218dac6e9570408b66962883dcb0f28
[4] https://info.uniswap.org/#/pools/0x88e6a0c2ddd26feeb64f039a2c41296fcb3f5640
[5] https://eigenphi.io/mev/eigentx/0x1e82ed1b04d0a0df667f64c7f341a9924c79465e84d9c10d265e988d0818e9c5
[6] https://eigenphi.io/mev/eigentx/0x95b1e7dc5f54a5f6ca02be2e17e26e2c73eccac374f88e7451691e88dfcd8fec
[7] https://eigenphi.io/mev/eigentx/0x95b1e7dc5f54a5f6ca02be2e17e26e2c73eccac374f88e7451691e88dfcd8fec?tab=block

4 Data and Empirical Analysis

The driving assumption in the theoretical analysis in Section 5 is that some builders are superior at extracting value from the top-of-block opportunities. In this section, we provide empirical evidence for this assumption. We use realized price-volatility on the CEX as a plausibly exogenous instrument that affects top-of-block but not block body opportunities. In particular, price movements on the CEX create arbitage opportunities since DEX prices are by design static until the next block. Large price movements create large arbitrage opportunities, small price movements create small arbitrage opportunities.

We obtained block-level data from Etherscan for a period corresponding to roughly a month from April 1st, 2023 to May 1st, 2023 (ETH blocks 16950609 to 17150609). We combined this data with detailed price data of ETHUSD from a leading centralized exchange (Binance) in the 12 seconds before each block was built. Price movement in this window gives us a rough estimate of the amount that can be earned through arbitrage with central exchanges for that block.

The merged data surfaces some clear patterns in builder-volatility relationships. Three builders – Manta, Rsync Builder, and Beaver Build – were identified before the analysis as likely to be better at extracting top of block MEV due to rumored connections with High Frequency Trading Firms. We show how realized volatility is related to whether or not one of these three builders constructed the block in Figure 1.

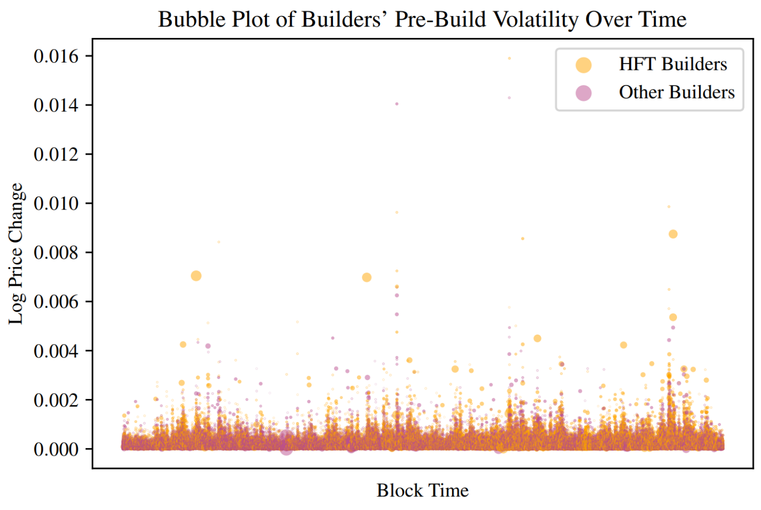


Figure 1 Blocks sorted by block time (x-axis) vs. pre-block volatility measured by log price change (y-axis).

When analyzing the most volatile blocks from each of the HFT traders, there are many large trades with Uniswap v3 pools at the top of each block (as we showed in Section 3). In general, the larger the realized volatility in the preceding 12 seconds, the larger these trades were. In some cases, when the realized volatility was most extreme, blocks included more than 30 CEX/DEX arbitrage transactions, with many consisting of notional sizes of millions of USD. We document blocks of several notable builders and their respective volatilities in Figure 2.

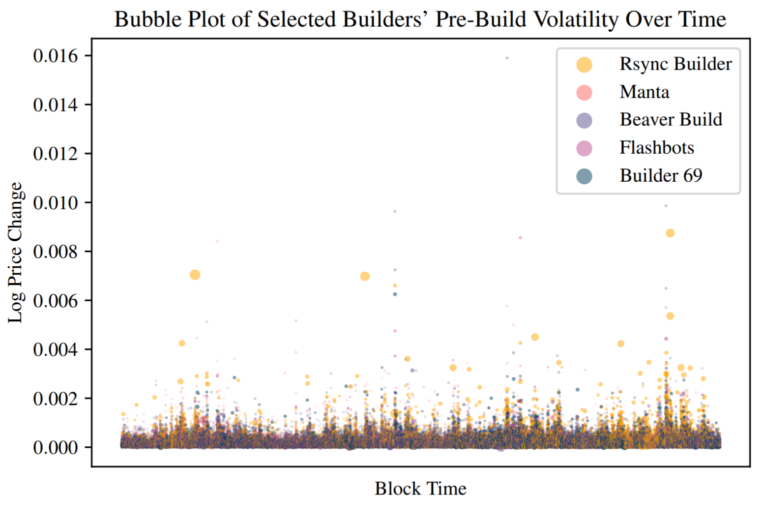


Figure 2 Selected builders' blocks sorted by block time (x-axis) vs. pre-block volatility measured by log price change (y-axis).

To formalize these findings, we model the relationship between CEX volatility and HFT builders winning the PBS auction. First, we grouped builders into HFT builders (Beaver, Manta, and Rsync) and non-HFT builders (everyone else). We regressed realized volatility on an indicator for whether or not one of these HFT builders won the block using a logistic regression:

$$P(\text{HFT Builder} = 1|\text{Log Price Change}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \cdot \text{Log Price Change})}}$$

Table 1 Logistic Regression Results.

	Coeff	**Std Err**	z	**P** $> \|z\|$	**[0.025**	**0.975]**
Log10 Price Change	2055.151***	(47.584)	43.190	0.000	1961.888	2148.414
const	-0.821***	(0.006)	-133.054	0.000	-0.833	-0.809

We find that the coefficient for the log price change predictor variable is 2055.151, with a standard error of 47.584. The significant positive relationship indicates that as the log price change increases, the odds of HFT builders winning the block also increases. Interpreting the model, when the Log10 Price Change is equal to 0 (i.e., no change) in the period before the block, the log odds of an HFT builder winning the block are -0.821. This corresponds to a probability of 0.306. If the realized volatility was 1% The probability that an HFT builder won the block was 0.775. When the realized volatility was 2% the probability that an HFT builder won the block was 0.964. Our analysis therefore comprehensively shows that the likelihood of the builders we preidentified as HFT builders winning the block grows as realized volatility increases. This suggests that these builders are much better than the rest of the field at extracting top-of-block value.

To identify differences between these HFT builder's capabilities, we construct a multinomial logistic regression:

$$\log\left(\frac{P(\text{Builder}_i)}{P(\text{Builder}_{\text{ref}})}\right) = \beta_{0i} + \beta_{1i}(\text{Log Price Change})$$

We restrict analysis to six builders: BeaverBuild, Blocknative, Builder 69, Flashbots, Manta, and Rsync Builder. Three of these (Beaver, Manta and Rsync) are our aforementioned HFT builders, the remaining three (Blocknative, Builder69 and Flashbots) are high-volume builders that construct a high percentage of the remaining blocks. This model analyzes how the CEX price volatility between blocks impacts the probability of one of these entities becoming the block winner. The resulting model coefficients for each builder in Table 2 estimate how a unit increase in Log Price Change before a block will impact the log ratio of the probability of that block being won by that particular builder vs. the probability of it being won by a builder in the reference class. While these coefficients are more difficult to interpret than simple logistic model with HFT builders, our findings show significant relationships between increased volatility before a block and that block being won by a particular builder: the coefficients are positive and significant for our preidentified HFT builders (Beaver, Manta and Rsync) , as one might have expected given our previous results. Conversely, they are significant and negative for the high-volume, non HFT builders (Builder69 and Flashbots).[8]

This suggests that these high-volume builders either do not compete in top-of-block extraction activity or at least are substantially less skilled relative to the HFT builders.

Table 2 MNLogit Regression Results.

	coef	**std err**	z	**P** $> \|z\|$	**[0.025**	**0.975]**
Beaver Build						
const	-0.4144	0.009	-45.929	0.000	-0.432	-0.397
Log10 Price Change	1386.2014	71.403	19.414	0.000	1246.254	1526.149
Blocknative						
const	-2.4772	0.020	-126.577	0.000	-2.516	-2.439
Log10 Price Change	1629.2443	138.304	11.780	0.000	1358.174	1900.315
Builder 69						
const	0.0152	0.008	1.799	0.072	-0.001	0.032
Log10 Price Change	-527.4993	75.762	-6.963	0.000	-675.991	-379.008
Flashbots						
const	-0.4522	0.010	-46.985	0.000	-0.471	-0.433
Log10 Price Change	-458.7271	86.446	-5.306	0.000	-628.159	-289.295
Manta						
const	-3.2312	0.023	-137.575	0.000	-3.277	-3.185
Log10 Price Change	3824.6414	104.548	36.583	0.000	3619.731	4029.551
Rsync Builder						
const	-0.6812	0.010	-71.400	0.000	-0.700	-0.662
Log10 Price Change	2093.8362	71.075	29.459	0.000	1954.532	2233.141

[8] We note that the coefficient is positive and significant for Blocknative even though Blocknative claims (and industry participants agree) that it is not an HFT builder. A possible reason for this is that Blocknative runs their own relay and presumably collocates their builder with their relay. This could give them a latency advantage which could be influential in winning high volatility blocks. Firms using HFT-type strategies searching for a latency edge may therefore use Blocknative, resulting in the strongly positive coefficient.

5 Model and Theoretical Analysis

Having demonstrated the core assumption (that some builders are better than others at extracting value from the top of the block), we turn to a theoretical model of what the downstream effects of this advantage might be as the prevalence of private order flow increases.

We construct and study a simple static model for a single slot. In our model, a block consists of at most 2 transactions. There is a single available block body transaction which can generate MEV (for example a swap transaction that can be sandwiched). Further, there is a single top-of-block CEX/DEX arbitrage opportunity. There are two builders, A and B. Each of these builders competes in the PBS auction to have their block included. In practice, the PBS auction is an English auction – we will simply consider the standard dominant strategy equilibrium of this auction, i.e., each agent stays in until their value, resulting in the highest value buyer winning at the second highest value.

We consider two scenarios. Scenario 1 models the current situation with little/ no private order flow, while scenario 2 models a setting with private order flow.

Scenario 1. In this setting the block body transaction is available to both builders, for example as a bundle from a third (unmodeled) searcher. Both builders therefore have the same value for this transaction, equaling the searcher's tip which is paid to the including builder – we will denote this value as v_T. At the time of the PBS auction, each builder $x \in \{A, B\}$ also sees their value v_x for the CEX/DEX arb. They then bid in the PBS auction, with the winning bidder's block being included.

Scenario 2. In this setting the block body transaction is available for sale at an OFA that runs prior to the PBS auction. The value of the transaction for sale is v_T, commonly known among the two bidders. For simplicity we will first assume that this auction runs as a second-price auction, i.e. builders submit bids and the winner (highest bid) pays the second-highest bid. In this setting, the loser of the auction does not have access to the block body transaction. At the time of the PBS auction, each builder $x \in \{A, B\}$ also sees their value v_x for the CEX/DEX arb. They then bid in the PBS transaction, with the winning bidder in this auction having their block included.

▶ **Assumption 1.** We will assume that for each $x \in \{A, B\}$, $v_x \sim F_x$ where F_x is a CDF on $[0, 1]$, and that $v_A \perp v_B$, i.e. A and B are independently drawn.

Further we assume that $F_A \succ_{\text{FOSD}} F_B$, i.e., builder A is stochastically better at CEX/ DEX arb than builder B.

Our results in this section show that the outcomes in Scenario 2, i.e., the scenario with OFAs and private order flow, overly advantage builder A over builder B relative to scenario 1.

5.1 Baseline Results

The basic idea is straightforward and can be easily described in a setting where v_A and v_B are deterministic (or equivalently, F_A and F_B are degenerate distributions). Without loss of generality, assume that $v_A > v_B$.

▶ **Theorem 2.** *In Scenario 1, suppose that v_x for each of $x \in A, B$ is common knowledge among the builders before bidding in the PBS auction. Then the equilibrium of the PBS auction is that A wins the PBS auction at price $v_T + v_B$. Their total profit is therefore $v_A - v_B$.*

In short, the Theorem asserts that the outcome in Scenario 1 allocates blockspace efficiently.

Proof. To see why, note that the block body transaction is available to both builders and has the same value, so the sole differentiation is in terms of their value for the top-of-block (CEX/DEX arb). The value of each bidder x for winning the auction is therefore $v_T + v_x$. In the standard equilibrium of an English auction with complete information, the outcome is efficient with the high value bidder winning at the second highest price. The theorem follows. ◀

As a first benchmark to compare this against, suppose in Scenario 2 the builders know their value for the CEX/DEX opportunity before the OFA begins.

▶ **Theorem 3.** *In Scenario 1, suppose that the value v_x for each builder $x \in A, B$ for top of block is common knowledge among them before bidding in the OFA. Then the overall outcome of OFA followed by PBS auction is that A wins both auctions at total price* $\max(v_T + 2v_B - v_A, v_B)$*. Their total surplus is therefore* $\min(2(v_A - v_B), v_A + v_T - v_B)$*.*

Proof. The proof follows straightforwardly from backward induction. We can work out the willingness to pay of each party for the transaction in the OFA based on the difference in profit in the PBS auction conditional on who wins the OFA. There are two mutually exclusive, totally exhaustive cases:

Case 1. $v_A > v_B + v_T$. In this case, note that the winner of the PBS auction is A regardless of who wins the OFA (since we already have that $v_A > v_B$). Therefore B gets a 0 surplus regardless. As a result, we have that B bids 0 in the OFA and therefore A wins the transaction. Then, the PBS auction clears at a price of v_B with A winning the block, and the total surplus of A is $v_A + v_T - v_B$.

Case 2. $v_A \leq v_B + v_T$. In this case, the winner of the OFA will go ahead and win the PBS (since the value of the transaction v_T plus their own value for the top-of-block opportunity combines will be larger than the competitor's value for the top-of-block opportunity). Note that if A wins the OFA, then they will therefore win the PBS at a price of v_B for a net surplus of $v_A + v_T - v_B$ (and B will make a total surplus of 0). Conversely, if B wins the OFA, they will win the PBS for a price of v_A, with a net surplus of $v_B + v_T - v_A$ (and A will make a total surplus of 0).

Therefore, A's willingness to pay for the transaction in the OFA is $v_A + v_T - v_B$, whereas B is willing to pay $v_B + v_T - v_A < v_A + v_T - v_B$ (since $v_A > v_B$ by assumption). As a result the OFA will see A winning for a price of $v_B + v_T - v_A$. Combining these (the outcomes of the PBA above and the OFA here) we have the desired result. ◀

These results already exhibit the "centralization effects" of private order flow on proposer builder separation: every additional dollar of advantage a builder has in top of block extraction translates into more than a dollars of surplus (for a small advantage, up to two dollars). In short, a builder who is already advantaged has a steeper incentive to invest in improving their advantage.

5.2 Stochastic Top-of-Block Opportunities

Our results carry through, *mutatis mutandis*, for a more realistic model where at the time of bidding in the OFA, builders do not know the value of the top of block opportunity. Of course this applies solely to Scenario 2. In this case, builder x at the stage of the OFA bids on the understanding that their top-of-block opportunity will be revealed to them later, and is distributed as $v_x \sim F_x$. At the conclusion of the OFA, the realized top-of-block opportunity for each builder is revealed to them, and is modeled as a private value.[9]

Suppose builder A wins the OFA. In this case, their value for the block is $v_T + v_A$, while builder B's value for the block is v_B. Conversely, if builder A loses the OFA, their value for the block is v_A while builder B's value for the block is $V_B + v_T$.

▶ **Theorem 4.** *Builder A's value for the transaction in the OFA, $v_{T,A}$, can be written as:*

$$v_{T,A} = \int_0^\infty \int_0^{v_A} F_B(v + v_T) - F_B(v - v_T) dv dF_A(v_A),$$

with $v_{T,B}$ defined analogously.

Proof. Note that conditional on builder A's value for top of block slot being v_A, their interim probability of winning the block is

$$x_A^{\text{win}}(v_A) = F_B(v_A + v_T),$$

and analogously their probability of winning the block from losing the OFA is

$$x_A^{\text{lose}}(v_A) = F_B(v_A - v_T).$$

Therefore, by the revenue equivalence theorem (see e.g., Proposition 3.1 of [6]), the expected surplus of builder A in the PBS auction, conditional on the outcome of the OFA with a value of V_A for the top of the block can be written as

$$s_A^{\text{win}}(v_A) = \int_0^{v_A} x_A^{\text{win}}(v) dv = \int_0^{v_A} F_B(v + v_T) dv,$$
$$s_A^{\text{lose}}(v_A) = \int_0^{v_A} x_A^{\text{lose}}(v) dv = \int_0^{v_A} F_B(v - v_T) dv$$

Finally, the ex-ante expected surplus from winning can be written as:

$$S_A^{\text{win}} = \int_0^\infty s_A^{\text{win}}(v_A) dF_A(v_A) = \int_0^\infty \int_0^{v_A} F_B(v + v_T) dv dF_A(v_A),$$

and expected surplus from losing as,

$$S_A^{\text{lose}} = \int_0^\infty s_A^{\text{lose}}(v_A) dF_A(v_A) = \int_0^\infty \int_0^{v_A} F_B(v - v_T) dv dF_A(v_A).$$

Therefore the effective valuation of builder A to win the the transaction in the OFA, $v_{T,A}$ equals $S_A^{\text{win}} - S_A^{\text{lose}}$. Analogously, the valuation of builder B in the transaction in the OFA equals $S_B^{\text{win}} - S_B^{\text{lose}}$.

[9] It maybe interesting to consider the case where this value is a signal of expected top-of-block value. In this case, we may be in a setting of interdependent values as in [8]. We leave that study to future work.

Note that

$$\begin{aligned} v_{T,A} &= S_A^{\text{win}} - S_A^{\text{lose}}, \\ &= \int_0^\infty \int_0^{v_A} F_B(v+v_T) - F_B(v-v_T) dv dF_A(v_A), \end{aligned}$$

and, analogously,

$$\begin{aligned} v_{T,B} &= S_B^{\text{win}} - S_B^{\text{lose}}, \\ &= \int_0^\infty \int_0^{v_B} F_A(v+v_T) - F_A(v-v_T) dv dF_B(v_A), \end{aligned}$$

as desired. ◀

Finally, note that under various assumptions, it can be shown that $v_{T,A} > v_{T,B}$. For example:

▶ **Corollary 5.** *Suppose v_T is small enough so that a Taylor series approximation is appropriate. Then $v_{T,A} \geq v_{T_B}$.*

Proof. To see this note that

$$\begin{aligned} v_{T,A} &= \int_0^\infty \int_0^{v_A} F_B(v+v_T) - F_B(v-v_T) dv dF_A(v_A), \\ &\approx \int_0^\infty \int_0^{v_A} 2v_T f_B(v) dv dF_A(v_A) \\ &= 2v_T \int_0^\infty F_B(v_A) f_A(v_A) dv_A. \end{aligned}$$

By an analogous argument,

$$v_{T,B} \approx 2v_T \int_0^\infty F_A(v_B) f_B(v_B) dv_B.$$

Since $F_A \succ_{\text{FOSD}} F_B$, we have that for all v, $F_A(v) \leq F_B(v)$. Therefore we have that,

$$\begin{aligned} v_{T,A} &\approx 2v_T \int_0^\infty F_B(v_A) f_A(v_A) dv_A \\ &\geq \int_0^\infty F_A(v_A) f_A(v_A) dv_A && (\text{since } F_A \succ_{\text{FOSD}} F_B), \\ &\geq \int_0^\infty F_A(v_A) f_B(v_A) dv_A && (\text{since } F_A \succ_{\text{FOSD}} F_B), \\ &\approx v_{T,B}. \end{aligned}$$ ◀

Note that this corollary already implies that even though the top of block opportunities are v_A and v_B are stochastic, builder A *always* wins the OFA, since it expects better (stochastic) top of block opportunities.

Using this, we can compare winning probabilities and builder profit across the two scenarios. We summarize our results with the following theorem:

▶ **Theorem 6.** *Under Scenario 1, builder A wins the block with ex-ante probability $\int_0^\infty F_B(v_A) f_A(v_A) dv_A$; whereas under OFAs with private transactions, builder A's winning probability increases to $\int_0^\infty F_B(v_A + v_T) f_A(v_A) dv_A$.*

Under Scenario 1*, the total expected profit of builder A is*

$$\int_0^\infty \int_0^{v_A} F_B(v)dvdF_A(v_A).$$

Under Scenario 2*, the total expected profit of builder A is*

$$(v_{T,A} - v_{T,B}) + \int_0^\infty \int_0^{v_A} F_B(v + v_T)dvdF_A(v_A).$$

Proof. To see the first part, note that under scenario 1, builder A wins the block whenever their realized value for the transaction (v_A) exceeds builder B's.

Under scenario 2, note that by the previous result, builder A always wins the block-body transaction in the OFA. It then therefore wins the PBS auction whenever builder B's value for the top-of-block transaction is at most v_T larger than builder A's value for the same. The formulae listed straightforwardly represent the probability of the events described above.

The total expected profits then follow from revenue equivalence (see e.g. Proposition 3.1 of [6]). In particular, recall that if a buyer of value wins the object with probability $x(v)$, their expected profit in this auction must be $S(v) = \int_0^v x(v')dv' + S(0)$. The expected profits listed above then follow straightforwardly. ◀

By observation, the profits of builder 1 have gone up: firstly, they make positive profit in the OFA since they are more aggressive in the OFA. Secondly, having won the OFA, they are advantaged in the PBS auction (since they have access to the private transaction to increase their value for the block, and builder B does not). Further results require us to make a functional form assumption on F_A and F_B, which we do in the next section.

5.3 An Analytic Example

To better understand the effect on surplus etc, we can use the formulas above in an analytic example so that we can do some simple comparative statics. To that end suppose both v_A and v_B are exponentially distributed, with parameter λ_A and λ_B respectively. By assumption that A is the stronger builder in terms of first order stochastic dominance of top-of-block opportunities, we must have that $\lambda_A < \lambda_B$.

Note that, for each $x \in \{A, B\}$

$$F_x(v) = 1 - \exp\{-\lambda_x v\},$$
$$f_x(v) = \lambda_x \exp\{-\lambda_x v\}.$$

Therefore, substituting in, we have that:

$$\begin{aligned} v_{T,A} &= \int_0^\infty \int_0^{v_A} F_B(v + v_T) - F_B(v - v_T)dvdF_A(v_A), \\ &= \int_0^\infty H_B(v_A)dF_A(v_A), \end{aligned}$$

where

$$H_B(v_A) = \begin{cases} \int_{v_T}^{v_A} F_B(v + v_T) - F_B(v - v_T)dv + \int_0^{v_T} F_B(v + v_T)dv & \text{if } v_A > v_T \\ \int_0^{v_A} F_B(v + v_T)dv & \text{o.w.} \end{cases}$$

A mechanical but involved calculation delivers that:

$$v_{T,A} = \frac{\lambda_A(1-\exp(-v_T\lambda_B)) + \lambda_B(1-\exp(-v_T\lambda_A))}{(\lambda_A^2 + \lambda_A\lambda_B)}$$

And analogously $v_{T,B}$. Further it is straightforward to verify that $v_{T,A} > v_{T,B}$ (since $\lambda_A < \lambda_B$ by assumption) as desired.

Substituting in to the formulas in Theorem 6, we have that the probability of A winning rises to

$$1 - \frac{\exp\{-v_T\lambda_B\}\lambda_A}{\lambda_A + \lambda_B} > \frac{\lambda_B}{\lambda_A + \lambda_B}$$

where the right hand side is the probability of A winning in Scenario 1.

Finally, note that under scenario 1, the total expected profit of Builder A is

$$\int_0^\infty \int_0^{v_A} F_B(v)dvdF_A(v_A) = \frac{\lambda_B}{\lambda_A(\lambda_A + \lambda_B)}.$$

By comparison, under scenario 2, the total expected profit of Builder A is:

$$(v_{T,A} - v_{T,B}) + \int_0^\infty \int_0^{v_A} F_B(v + v_T)dvdF_A(v_A),$$
$$= \frac{(\lambda_B - \lambda_A)(\lambda_A(1-\exp(-v_T\lambda_B)) + \lambda_B(1-\exp(-v_T\lambda_A)))}{\lambda_A\lambda_B(\lambda_A + \lambda_B)} + \frac{\lambda_B + \lambda_A(1-\exp(-v_T\lambda_B))}{\lambda_A(\lambda_A + \lambda_B)}.$$

Therefore the difference in profit between the two scenarios is:

$$\frac{(\lambda_B - \lambda_A)(\lambda_A(1-\exp(-v_T\lambda_B)) + \lambda_B(1-\exp(-v_T\lambda_A)))}{\lambda_A\lambda_B(\lambda_A + \lambda_B)} + \frac{(1-\exp(-v_T\lambda_B))}{(\lambda_A + \lambda_B)}$$

Note that since each of the terms is positive, so is the sum, i.e. builder A's total profit increases in Scenario 2 relative to scenario 1.

These comparative statics are illustrated in Figure 3. We normalize v_T to 1, and capture the advantage of builder A by varying $\frac{\lambda_A}{\lambda_A+\lambda_B}$ holding $\lambda_A + \lambda_B$ fixed. The smaller the former, the larger is builder A's advantage in top of block extraction. The figure threfore demonstrates how even small advantages can be discontinuously magnified by private OFAs: it is instructive to note that even if the advantaged builder has a small advantage in top-of-block extraction, e.g., $\lambda_B = \lambda_A + \epsilon$ for ϵ small, they have a discontinuous jump in their probability of winning the PBS auction in scenario 2 relative to scenario 1. This is because even a small advantage in top-of-block extraction leads to the advantaged builder always winning the OFA in Scenario 2, which in turn gives them a discontinuous advantage in the PBS auction.

▶ Remark 7 (Bundle Sharing). The results of this model apply in a world where builders do not share opportunities with other builders. This practice, known as bundle sharing, is rather common [14]. When a bundle is shared, the bundle tip sets how much of the opportunity's value is shared with the builder and how much is retained for the bundle originator. In a world with low bundle sharing friction, the centralizing effects of private order flow are diminished because the value of running a builder is low (searchers can get almost the same execution without a builder as they can if they do run a builder).

In practice there are several frictions that make bundle sharing less desirable. First, a builder sharing a bundle with another searcher can cause the competing builder to elevate their bid in this PBS auction, making it more likely that the originating builder loses the

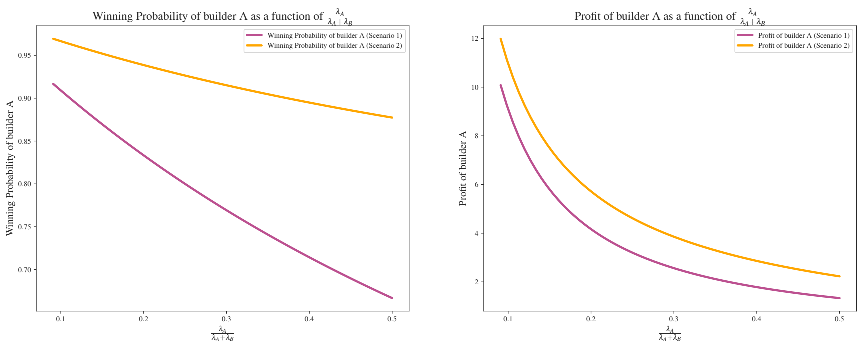


Figure 3 How winning probability (left) and expected profit (right) vary across scenarios 1 and 2 as the relative advantage of Builder A varies.

PBS auction or pays a higher price when he wins. Second, bundles submitted to other builders have higher latency, meaning decisions have to be made earlier in the slot with less information about how the prices of underlying assets will evolve. This latency effect is particularly relevant for top of block arbitrage opportunities which is why most successful top of block searchers also run their own builder.

6 Discussion

Our empirical results show that a small group of integrated builder-searchers have a demonstrable advantage in top-of-block extraction capability.

Our theoretical model then shows that builders with superior top-of-block capabilities are likely to dominate OFAs and subsequently use the private order flow obtained in these OFAs to dominate the PBS auction. Put simply, top-of-block and block-body opportunities are complementary because the block is sold wholesale. Therefore, builders who earn more from the top of the block, will be willing to pay more for private order flow, since they need to win the whole block in order to exercise their top-of-block advantage. This complementarity is a strong centralizing force that threatens to suffocate small builders and upset the currently somewhat pluralistic builder equilibrium.

Asking order flow originators not to participate in OFAs is futile because it is in their own best interest to do so. Similarly, builders cannot be barred from participating in OFAs. The only solution then is to modify PBS itself.

Our results suggest that unbundling the PBS auction would be a step in the right direction. By this we mean selling the top of the block and the block-body separately. Implementing such a mechanism would reduce HFT advantage and allow alternative strategies to integrated builder searchers to compete for the right to build blocks. A more fleshed-out proposal in this direction was recently proposed in [12] in the form of PEPC-Boost, a specific instantiation of a protocol enforced proposer commitment (PEPC) in which the block is split into designated top-of-block and blockbody and these are auctioned separately.

References

1 Vitalik Buterin. How much can we constrain builders without bringing back heavy burdens to proposers? https://ethresear.ch/t/how-much-can-we-constrain-builders-without-bringing-back-heavy-burdens-to-proposers/13808, October 2022.

2 Stephane Gosselin, Ankit Chiplunkar, and ConsenSys Research. The orderflow auction design space. *Frontier Research*, 6:83, 2023. URL: https://frontier.tech/the-orderflow-auction-design-space/.

3 Lioba Heimbach, Lucianna Kiffer, Christof Ferreira Torres, and Roger Wattenhofer. Ethereum's proposer-builder separation: Promises and realities, 2023. arXiv:2305.19037.

4 Quintus Kilbourn. Order flow, auctions, and centralization. *The Flashbots Collective*, 2022. URL: https://writings.flashbots.net/order-flow-auctions-and-centralisation.

5 Quintus Kilbourn. Order flow, auctions, and centralization ii. *Flashbots*, 2022. URL: https://writings.flashbots.net/order-flow-auctions-and-centralisation-II.

6 Vijay Krishna. *Auction theory*. Academic press, 2009.

7 Julian Ma. Structuring blockspace derivatives. https://mirror.xyz/0x03c29504CEcCa30B93FF5774183a1358D41fbeB1/WKa3GFC03uY34d2MufTyD0c595xVRUEZi9RNG-dHNKs, October 2022.

8 Paul R Milgrom and Robert J Weber. A theory of auctions and competitive bidding. *Econometrica: Journal of the Econometric Society*, pages 1089–1122, 1982.

9 Jason Milionis, Ciamac C. Moallemi, and Tim Roughgarden. Automated market making and arbitrage profits in the presence of fees, 2023. arXiv:2305.14604.

10 Jason Milionis, Ciamac C. Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Automated market making and loss-versus-rebalancing, 2022. arXiv:2208.06046.

11 Barnabe Monnot. Unbundling pbs: Towards protocol-enforced proposer commitments (pepc). https://ethresear.ch/t/unbundling-pbs-towards-protocol-enforced-proposer-commitments-pepc/13879?u=barnabe, October 2022.

12 Barnabé Monnot. Pepc faq. https://efdn.notion.site/PEPC-FAQ-0787ba2f77e14efba771ff2d903d67e4, July 2023.

13 Caspar Schwarz-Schilling, Fahad Saleh, Thomas Thiery, Jennifer Pan, Nihar Shah, and Barnabé Monnot. Time is money: Strategic timing games in proof-of-stake protocols, 2023. arXiv:2305.09032.

14 Titan. Builder dominance and searcher dependence. *Frontier Tech*, 2023. URL: https://frontier.tech/builder-dominance-and-searcher-dependence.

15 Anton Wahrstätter, Liyi Zhou, Kaihua Qin, Davor Svetinovic, and Arthur Gervais. Time to bribe: Measuring block construction market, 2023. arXiv:2305.16468.

When Bidders Are DAOs

Maryam Bahrani
a16z Crypto, New York, NY, USA

Pranav Garimidi
a16z Crypto, New York, NY, USA

Tim Roughgarden
a16z Crypto , New York, NY, USA
Columbia University, New York, NY, USA

Abstract

In a typical decentralized autonomous organization (DAO), people organize themselves into a group that is programmatically managed. DAOs can act as bidders in auctions (with ConstitutionDAO being one notable example), with a DAO's bid typically treated by the auctioneer as if it had been submitted by an individual, without regard to any details of the internal DAO dynamics.

The goal of this paper is to study auctions in which the bidders are DAOs. More precisely, we consider the design of two-level auctions in which the "participants" are groups of bidders rather than individuals. Bidders form DAOs to pool resources, but must then also negotiate the terms by which the DAO's winnings are shared. We model the outcome of a DAO's negotiations through an aggregation function (which aggregates DAO members' bids into a single group bid) and a budget-balanced cost-sharing mechanism (that determines DAO members' access to the DAO's allocation and distributes the aggregate payment demanded from the DAO to its members). DAOs' bids are processed by a direct-revelation mechanism that has no knowledge of the DAO structure (and thus treats each DAO as an individual). Within this framework, we pursue two-level mechanisms that are incentive-compatible (with truthful bidding a dominant strategy for each member of each DAO) and approximately welfare-optimal.

We prove that, even in the case of a single-item auction, the DAO dynamics hidden from the outer mechanism preclude incentive-compatible welfare maximization: No matter what the outer mechanism and the cost-sharing mechanisms used by DAOs, the welfare of the resulting two-level mechanism can be a $\approx \ln n$ factor less than the optimal welfare (in the worst case over DAOs and valuation profiles). We complement this lower bound with a natural two-level mechanism that achieves a matching approximate welfare guarantee. This upper bound also extends to multi-item auctions in which individuals have additive valuations. Finally, we show that our positive results cannot be extended much further: Even in multi-item settings in which bidders have unit-demand valuations, truthful two-level mechanisms form a highly restricted class and as a consequence cannot guarantee any non-trivial approximation of the maximum social welfare.

2012 ACM Subject Classification Applied computing → Online auctions

Keywords and phrases Auctions, DAOs

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.21

Funding *Tim Roughgarden*: Research at Columbia University supported in part by NSF awards CCF-2006737 and CNS-2212745.

Acknowledgements We thank Dan Boneh for helpful discussions and the reviewers for comments on future directions.

1 Introduction

In November 2021, one of the 13 extant original copies of the U.S. Constitution was put up for sale by Sotheby's auction house. Within days, a DAO ("decentralized autonomous organization") with over 17000 members formed to crowdsource funds to participate in the auction. This DAO – called ConstitutionDAO, naturally – believed that physical copies of the Constitution should be controlled "by the people," rather than stuck in private collections. Members of this DAO valued belonging to the collective that wins this auction, and from their perspective the good for sale is therefore non-rivalrous among them, with a member's value for winning unharmed by the fact that other members (of the same DAO) win as well. The members of the DAO were generally anonymous; committed funds were publicly visible and held in escrow in a smart contract deployed to the Ethereum blockchain. All told, the DAO raised roughly $47 million leading up to the auction. Sotheby's sold the copy of the Constitution using (of course) an ascending auction, with a designated DAO representative relaying the bids implied by the DAO's reserves. Thus, from Sotheby's perspective, ConstitutionDAO was just like any other bidder, even though in reality it represented the outcome of coordination of thousands of individuals. (In the end, ConstitutionDAO lost the auction to Ken Griffin, CEO of the hedge fund Citadel, and the escrowed funds were returned to the DAO's participants.)

More generally, the point of a DAO is for people to organize themselves into groups that are programmatically managed. Usually these DAOs are centered around some kind of common cause, for example, forming a social club, collecting art, or funding projects. The key innovation that DAOs bring over traditional collectives is that the behavior of the DAO is programmatically enforced via blockchain-secured smart contracts, allowing DAOs to use more complex mechanisms than would traditionally be possible. These DAOs may find themselves competing in auctions on behalf of their members. In a typical such auction, bids by DAOs might be treated by the seller as individual bids in (say) a first-price auction, without regard to any details of the internal group dynamics.

Lest ConstitutionDAO seem like an isolated example, we stress that as blockchains and DAOs become mainstream, this same pattern will likely recur. For example, a DAO concerned with environmental activism could compete in an auction to buy the right to preserve a certain area of land. A DAO of musicians could compete for the long-term use of a particular performance venue. In many of these settings, as long as you are part of the winning DAO, you have a value for the good that is independent of how many other people are part of the DAO. Depending on the application, the good may or may not be excludable (i.e., with the option of excluding select DAO members from access); as our results show, the degree to which the good for sale is excludable will have a first-order effect on whether there are mechanisms with good incentive and welfare guarantees.

The goal of this paper is to study auctions in which the bidders are DAOs. Obvious questions then include: How should DAO dynamics and internal negotiations be modeled? Which key lessons of classical auction theory hold also when bids represent DAOs, and which ones must be revisited? Do the mechanism design problems at the "upper level" (the choice of auction) and the "lower level" (the aggregation of preferences of members of a DAO) compose nicely (*e.g.*, preserving incentive-compatibility), or are there intricate interactions between them? Does the aggregation of multiple individual preferences into a single DAO preference interfere with natural mechanism design objectives like welfare-maximization, and if so, by how much? This paper initiates the study of these questions.

1.1 Informal Description of Our Model

Section 2 details our model; here, we provide an informal description that is sufficient to understand the overview of results in the next section.

First, there is an auctioneer that runs an *upper-level* mechanism, which takes as input bids from groups (representing DAOs), and outputs an allocation of items to groups along with group payments. This mechanism has no knowledge of (or is deliberately designed to ignore) the process by which groups' bids were produced; for all the mechanism knows, each bid was submitted by an individual bidder. First- and second-price single-item auctions are canonical examples of such mechanisms.

Given the output of the upper-level mechanism, each group must determine each member's access to then group's winnings, and what to charge each member to cover the overall payment demanded by the auctioneer. In the spirit of the Revelation Principle, we model the result of these determinations as a choice of a (direct-revelation) budget-balanced cost-sharing mechanism, where the cost to be shared is the payment demanded by the auctioneer. In addition to choosing this *lower-level* mechanism, a group must decide what to bid (in the upper-level mechanism), as a function of its members' bids. We refer to this mapping (from members' bids to a single group bid) as an *aggregation function*.

Summarizing, given a choice of upper-level and lower-level mechanisms (including the aggregation functions), the overall sequence of events unfolds as follows: (i) each member of each group submits an individual bid to that group's lower-level mechanism; (ii) each group maps its members' bids to a group bid via its aggregation function, which is then submitted to the upper-level mechanism; (iii) the upper-level mechanism chooses, as a function of the submitted group bids, an allocation of its items to groups and payments by the groups in exchange for the allocated items; (iv) each group, having received its items and a payment request from the upper-level mechanism, uses its lower-level mechanism to give its members access to the items won and to share the payment among its members.

Within this framework, we pursue two goals: (i) dominant-strategy incentive-compatibility (DSIC); and (ii) (approximate) social welfare maximization. By DSIC, we mean a two-level mechanism in which every member of every group has a dominant strategy, and that strategy is to bid its true valuation. For welfare, we consider the valuation of each group member for the subset of items (its group won and) to which it has access. The social welfare is the sum of these quantities over all members of all groups.

The core difficulty of this mechanism design problem is aggregating the preferences of a group and charging payments in an incentive-compatible way. For example, if a group always grants all its members access to all its items and shares the cost evenly, budget-balanced incentive-compatibility is impossible (due to free riders underbidding in the hopes that other group members will shoulder the cost of acquiring a valuable item). For this reason, two-level mechanisms with non-trivial guarantees must use lower-level mechanisms that can exclude group members – presumably the lower-bidding ones – from accessing some of the items allocated to the group. For instance, if a DAO of musicians wins access to a performance venue, it may designate a subset of DAO members – intuitively, the members whose bids were actually used to cover the cost of winning the auction – as the only ones eligible to book concerts at the venue.

A second challenge is that distributing payments within a group in an incentive-compatible way generally precludes the group from covering payments that match the full welfare of its members.

We prove that this challenge necessarily causes information to be lost in the aggregation process, which leads to an unavoidable indistinguishability problem for the upper-level mechanism and a consequent loss in social welfare.

1.2 Summary of Results

We begin with the canonical setting of a single-item auction, and identify a natural incentive-compatible two-level mechanism that guarantees an H_ℓ-approximation to the social welfare, where ℓ denotes the maximum number of bidders in any group and $H_\ell = \sum_{i=1}^{\ell} \frac{1}{i} \approx \ln \ell$ the

ℓth Harmonic number. The rough idea is that each group bids the maximum amount that can be shared equally among a subset of its members (subject to individual rationality), with a Vickrey (*i.e.*, second-price) auction serving as the upper mechanism. We complement this upper bound with a matching negative result, assuming only a weak "equal treatment" property.[1] Precisely, every incentive-compatible and individually rational two-level mechanism that satisfies this property cannot guarantee a worst-case approximation factor smaller than H_ℓ. This lower bound arises from the inability of truthful mechanisms to elicit payments from groups that match the groups' true welfare when members of a group have very unequal values.

We then proceed to the multi-item setting. The mechanism of our positive result for the single-item case extends easily and without degradation to the setting in which each member of each group has an additive valuation over items. However, the story changes dramatically when we consider the other canonical "easy case" for multi-item settings, namely bidders with unit-demand valuations. Here, we show that no incentive-compatible and individually rational two-level mechanism can achieve a better-than-n approximation of the optimal social welfare (where n denotes the total number of participants). The high-level idea of our proof is to show that incentive-compatibility is possible in this setting only if there are instances that require the mechanism to allocate all the items to a single group. We then prove that, because the upper mechanism is oblivious to the group structure (*e.g.*, whether a group represents a single bidder or many), there will be instances in which such allocations lead to extremely poor welfare. This negative result shows that, in particular, the composition of an incentive-compatible and approximately welfare-maximizing upper level mechanism (such as the VCG mechanism) with incentive-compatible and approximately welfare-maximizing lower mechanisms (such as maximum equal-split cost-sharing) need not lead to a two-level mechanism with those same properties. The design of two-level mechanisms with good provable guarantees thus requires careful coordination between the upper and lower mechanisms, along with strong restrictions on the set of feasible allocations or the structure of bidders' preferences.

1.3 Related Work

This paper follows in the tradition of a long line of works, beginning with [14] and [8], that study the problem of incentive-compatible approximate welfare-maximization under side constraints. (Without side constraints, exact incentive-compatible welfare-maximization can be achieved using the VCG mechanism.) Like most of these works, we focus on a prior-free setting, worst-case (over valuation profiles) relative approximation of the optimal welfare, and dominant-strategy incentive-compatibility.

Many of the papers in this line belong to the field of algorithmic mechanism design, in which the side constraints impose bounds on the amount of computation or communication used by a mechanism (see e.g. [19]). For example, in multi-item (combinatorial) auctions, the size of a bidder's valuation (and hence the communication used by a direct-revelation mechanism) is generally exponential in the number of items m. If a mechanism uses an amount of communication that is bounded by a polynomial function of m, bidders will be unable to report fully their valuations and the mechanism must ultimately make its allocation (and

[1] This property states that if two members of a group submit identical bids, they should also receive identical allocations (with either both or neither granted access to the item) and make identical payments. This is effectively a symmetry condition on how a mechanism breaks ties, and it is relevant only for a measure-zero set of valuation profiles (those in which some valuation is repeated).

payment) decisions with incomplete information. Unsurprisingly, full welfare-maximization is generally impossible in such settings, even after setting aside any incentive-compatibility constraints.

Somewhat similarly, in the two-level mechanism framework studied in this paper, one of the side constraints requires the upper mechanism to base its allocation (and payment) decisions on incomplete information (group bids, rather than the individual member bids that led to those group bids), again precluding any direct-revelation solution. Here, however, it is the combination of limited information *and* the incentive-compatibility constraint that rules out exact welfare-maximization.[2] Accordingly, the crux of our lower bound proofs is to delineate the limitations of incentive-compatible (two-level) mechanisms, not to identify any intrinsic difficulty of the underlying optimization problem. Several papers in algorithmic mechanism design, such as [15] and [4], face similar challenges when proving that there are welfare-maximization problems for which incentive-compatibility constraints degrade the best-possible approximation factor achievable by a polynomial-time algorithm. Results in the spirit of Roberts's Theorem [17], which characterize the set of incentive-compatible mechanisms for a given setting, can also have a similar flavor.

Our model is similar to the one in [16], although that paper has very different goals than the present work. Rather than considering the space of dominant-strategy incentive-compatible mechanisms, as we do here, the paper [16] fixes specific (non-incentive-compatible) auctions for the upper mechanism and aggregation rules and cost-sharing rules in the lower mechanism before characterizing (prior-dependent) equilibrium strategies for the participants. The analysis in [16] is also restricted to the specific setting in which a single group of bidders competes with a single individual bidder.

The lower mechanisms in our two-level framework are required to be budget-balanced cost-sharing mechanisms, and there is a large literature on such mechanisms. Naturally, some ideas in our proofs also have precursors in that literature; a few other papers that bear resemblance to the present work are [5], [13], [18], and [3]. There are two major differences, however, between the use of cost-sharing mechanisms in our framework and the settings in which they are traditionally studied. First, in the standard setup, a cost-sharing mechanism chooses an outcome that incurs a cost (*e.g.*, the cost of building a bridge) and the goal is to maximize the social welfare (the total value of the winning participants for the chosen outcome, minus the cost of that outcome) or minimize the social cost (the cost of the chosen outcome, plus the total value of the losing participants). The cost of an outcome in this setup is exogenously specified, independent of participants' bids. In our two-level framework, the "cost" to be shared is an endogenously specified transfer (from a group to the auctioneer, as a function of other groups' bids) that does not detract from the social welfare. In the traditional setup, incentive-compatible budget-balanced cost-sharing mechanisms cannot guarantee any approximation of the optimal social welfare for even the simplest of problems, and for this reason the social cost objective is usually considered instead [18]. Here, with costs internal rather than external to the system, a non-trivial approximation to the social welfare objective is possible (*e.g.*, for single-item settings). Second, cost-sharing mechanisms are traditionally studied as stand-alone direct-revelation mechanisms, whereas here they constitute one component of a more complex mechanism. Our results show that

[2] *E.g.*, in a single-item setting, exact welfare maximization (without incentive-compatibility) is easy to achieve: bidders bid truthfully, each group reports the sum of its members' bids, the upper mechanism chooses the highest group bid and charges that group its bid, and the winning group then charges all its members their bids.

plugging incentive-compatible cost-sharing mechanisms into a two-level mechanism with an incentive-compatible upper mechanism does not generally preserve incentive-compatibility (see Appendix A). This lack of modularity between the upper and lower mechanisms suggests that the power and limitations of two-level mechanisms must be studied from first principles.

Resembling our two-level framework is a sequence of papers on bidding rings and collusion in auctions, namely [7], [11], [9], [10], and [12]. In this line of work, a group of bidders participates in a first- or second-price single-item auction by coordinating among themselves in a *bidding ring* to increase their expected utility. Unlike in our model, in which individuals value belonging to the winning group (and hence many can "win"), in these papers there is only one winning individual. As a result, these works require transfers within the bidding ring to incentivize agents to join. They also require individuals to have a common prior and compute (non-dominant) equilibrium strategies. Finally, these works do not consider the welfare loss due to collusion, as we do here.

Also reminiscent of our two-level framework but more distantly related are works that consider various mechanism design setups with intermediaries. For example, [2] consider facility location problems on trees and assume that strategic agents report to mediators that then act on their behalf. In addition to studying a very different underlying optimization problem, a key aspect of this paper is that mediators are assumed to be strategic (whereas the analog in our framework, the lower mechanisms, have no agency). A related line of research, motivated by online advertisement exchange systems, considers auctions in which bidders report bids to intermediaries who in turn submit bids to a seller (*e.g.*, [6] and [1]). In addition to focusing on strategic intermediaries, these works are primarily concerned with approximate revenue-maximization (as opposed to approximate welfare-maximization).

2 Preliminaries

We consider a setting where an auctioneer is selling a set of items, $[m] = \{1, ..., m\}$ to k distinct groups. We denote group j by G^j and let G^j have n_j bidders, with a total of n bidders across all of the groups. We only allow bidders to be part of a single group and assume that the auctioneer only interacts with a group as a whole, with no insight into the inner group structure. We further assume the auctioneer is a trusted party who will follow the mechanism as specified.[3]

Each item l is constrained to being allocated to a single group but there are no constraints on how many bidders within that group can have access to the item. In other words, given that the auctioneer allocates l to G^j, G^j has full autonomy on deciding what subset of its members get allocated (*i.e.*, granted access to) item l. In this sense, a group treats an allocated item as a public excludable good.

We will refer to the ith bidder in group j as bidder i^j. Each bidder has a valuation function for the items they receive $v_i^j : \mathcal{P}([m]) \to \mathbb{R}_+$. We assume bidders have quasi-linear utilities where if bidder i^j is allocated a set of items S_i^j and pays p_i^j then $u_i^j(S_i^j, p_i^j) = v_i^j(S_i^j) - p_i^j$. Let $\mathcal{B}$ denote the bidding language for bidders. $\mathcal{B}$ will always be expressive enough for bidders to express their true valuation function. Each bidder acts strategically to submit a bid $b_i^j \in \mathcal{B}$ to their group seeking to maximize their utility.

We consider two main classes of valuation functions for bidders in this work:

[3] In the context of DAOs, the lower-level mechanism and aggregation functions can be run programmatically on a smart contract, eliminating the need to actually appoint a trusted auctioneer.

Additive. Additive bidders have a value for each item and their value for a set of items is the sum of their values for the individual items in that set. Formally, for each of the items $l \in [m]$ each bidder i^j has some value $v_i^j(l) \in \mathbb{R}_+$. Then bidder i^j's value for a set $S \subseteq [m]$ of items is $v_i^j(S) = \sum_{l \in S} v_i^j(l)$.

Unit-Demand. Unit-demand bidders will have a value for each item but their value for a set of items will only be the highest value they have for any item in that set. Formally, for each of the items $l \in [m]$ each bidder i^j has some value $v_i^j(l) \in \mathbb{R}_+$. Then i^j's value for a set $S \subseteq [m]$ of items is $v_i^j(S) = \max_{l \in S} v_i^j(l)$.

In both cases of valuation functions, the bidding language $\mathcal{B}$ consists of vectors $b_i^j \in \mathbb{R}_+^m$ where $b_i^j(l)$ specifies bidder i^j's bid for item l.

We define a two-level mechanism as consisting of two parts, a lower and upper mechanism. The upper mechanism is run by the auctioneer and takes as input bids from each of the groups. The auctioneer then decides which items should be allocated to which groups and how much those groups should pay. The upper mechanism falls into the typical mechanism design framework. The lower mechanism is run by each group and dictates how the group should aggregate bids from its members into a group bid. Then, given an allocation of items and a payment request from the auctioneer, the lower mechanism specifies how a group should assign items to bidders in the group and how much each bidder should pay to cover the group's payment to the auctioneer. In this work we will only consider deterministic mechanisms of this form.

Formally, a lower mechanism $\mathcal{M}_l = (a, x^l, c)$ for a group with n bidders consists of:

- An *aggregation rule* $a : \mathcal{B}^n \to \mathcal{B}$ mapping a vector of member bids into a single bid for the group.
  We insist that a is the identity function when $n = 1$; Intuitively, if the upper mechanism and lower mechanisms are truthful, there is no reason for the aggregation function to distort the bid of a bidder in a group of size one.
- A (lower) *allocation rule* $x^l : \mathcal{B}^n \times \mathcal{P}([m]) \times \mathbb{R}_+ \to A^l$. Each element of A^l specifies which items each bidder has access to based on which items the group is allocated and how much the group has to pay. We will sometimes refer to a specific lower allocation by $x^l = (x_1^l, ..., x_{n_j}^l)$ where each x_i^l returns which set of items i^j is allocated.
- A *cost-sharing rule* $c : \mathcal{B}^n \times \mathcal{P}([m]) \times \mathbb{R}_+ \to \mathbb{R}_+^n$ that specifies how much each bidder in the group has to pay based of which items they are given access to and how much the overall group has to pay. c_i^j will be the function that specifically denotes how much bidder i^j has to pay.

An upper mechanism $\mathcal{M}_u = (x^u, p)$ with k distinct groups consists of:

- An (upper) *allocation rule* $x^u : \mathcal{B}^k \to A^u$ where each element of A^u specifies which items each group is allocated. We constrain the allocation rule such that the mechanism can allocate each item to at most 1 group. We will sometimes refer to a specific upper allocation by $x^u = (x_1^u, ..., x_k^u)$ where each x_j^u returns which set of items group G^j is allocated.
- A *payment rule* $p : \mathcal{B}^k \to \mathbb{R}_+^k$ specifying how much the upper mechanism charges each group. p^j will be the function that specifically denotes how much group G^j has to pay. We sometimes abuse notation and use p^j to also refer to the specific amount group j has to pay in a particular instance.

We can now formally define a two-level mechanism.

▶ **Definition 1.** *A two-level mechanism $\mathcal{M} = (\mathcal{M}_u, \mathcal{M}_l)$ is defined by a pair of lower and upper mechanisms.*

An allocation of items to bidders is $\{S_i^j\}_{i,j}$. This implies an allocation of $S^j = \cup_{i=1}^{n_j} S_i^j$ to each group. We say an allocation is feasible if $S^{j_1} \cap S^{j_2} = \emptyset$ for all $j_1, j_2 \in [k]$ where $j_1 \neq j_2$. This implies that an allocation is feasible as long as no item is allocated to more than one group.

The social welfare of an allocation is given by $\sum_{j=1}^{k} \sum_{i=1}^{n_j} v_i^j(S_i^j)$. The optimal social welfare for an instance I, $\texttt{OPT}(I)$ is the maximum social welfare obtainable over feasible allocations given the valuation functions specified by I.[4] We refer to the social welfare a mechanism $\mathcal{M}$ achieves in some instance I by $SW(\mathcal{M}(I))$. We say that a mechanism achieves an α approximation to optimal welfare if for every instances I, we have that $\alpha \geq \frac{\texttt{OPT}(I)}{SW(\mathcal{M}(I))}$.

Given this setup we seek the following properties from any two-level mechanism:

Incentive-Compatibility. A mechanism $\mathcal{M}$ is incentive compatible if for all instances, each bidder has a dominant strategy to report their true valuation as their bid. More formally, if v_{-i}^j denote the values of all bidders apart from i^j, then $u_i^j(v_i^j, v_{-i}^j) \geq u_i^j(\tilde{v}_i^j, v_{-i}^j)$ for any $\tilde{v}_i^j \in V, v_{-i}^j \in V^{n-1}$ where V is the set of possible valuations for any bidder.

Budget-Balance. A mechanism $\mathcal{M}$ is budget balanced if for every group, the payment by a group's members exactly covers the cost charged by the auctioneer, $\sum_{i=1}^{n_j} p_i^j = p^j \ \forall j \in [k]$.

Individual Rationality. A mechanism $\mathcal{M}$ is individually rational if bidders that bid truthfully always have non-negative utility. That is, for any bidder i^j that bids truthfully, if $\mathcal{M}$ outputs an allocation $\{S_i^j\}_{i,j}$ and prices $\{p_i^j\}_{i,j}$, then $v_i^j(S_i^j) \geq p_i^j$.

The following properties (of a two-level mechanism) will also be important for some of our lower bound proofs:

Equal Treatment. A mechanism $\mathcal{M}$ satisfies equal treatment if any two bidders in the same group i_1^j, i_2^j that make the same bids also receive the same allocation and the same payment. That is, if $b_{i_1}^j = b_{i_2}^j$, then the allocation $\{S_i^j\}_{i,j}$ and prices $\{p_i^j\}_{i,j}$ output by $\mathcal{M}$ must satisfy $S_{i_1}^j = S_{i_2}^j$ and $p_{i_1}^j = p_{i_2}^j$.

Consumer Sovereignty. A mechanism $\mathcal{M}$ satisfies consumer sovereignty if a bidder can force the mechanism to allocate it a specific bundle by bidding sufficiently high. Formally, for any bidder i^j, given bids by all other bidders b_{-i}, there exists some bid b_i^j such that $\mathcal{M}$ outputs an allocation $\{S_i^j\}_{i,j}$ where $v_i^j(S_i^j) = \max_{S \subseteq [m]} v_i^j(S)$.

Upper Semi-Continuity. A mechanism $\mathcal{M}$ satisfies upper semi-continuity if for any bidder i^j, given bids b_{-i} by the outputs an allocation where $v_i^j(S_i^j) = \max_{S \subseteq [m]} v_i^j(S)$ other bidders, if i^j is allocated S_i^j for all bids $\tilde{b}_i^j \neq b_i^j$ with $\tilde{b}_i^j \geq b_i^j$ (component-wise over items), then i is allocated S_i^j by bidding b_i^j as well.

[4] In any optimal allocation, every group might as well give all of its allocated items to all of its members.

3 Single-Item Mechanisms

We begin our investigation of two-level mechanisms in the canonical setting of single-item auctions. The item can be allocated to one group and that group can grant access to the item to any subset of its members. Even in this simple setting, we prove that no incentive-compatible mechanism can achieve a constant-factor approximation of the optimal social welfare. Instead, we provide a mechanism that achieves a $H_n \approx \ln n$ approximation and show that this is the best any truthful mechanism can do.

Denote the single item by g. Then every bidder i^j has a value $v_i^j \in \mathbb{R}_+$ if they are allocated the item and value 0 otherwise. The bidding language is $\mathcal{B} = \mathbb{R}_+$.

3.1 Truthful Mechanism

We propose a two-level mechanism that is truthful, budget-balanced, individually rational, and satisfies equal treatment while obtaining a H_ℓ-approximation of the optimal social welfare (where ℓ denotes the largest number of members of any group and $H_\ell = \sum_{i=1}^{\ell} 1/i$ the ℓth Harmonic number).

Since $b_i^j \in \mathbb{R}_+$, assume without loss of generality that for each group G^j, $b_1^j \geq b_2^j \geq \ldots \geq b_{n_j}^j$. The upper allocation is represented by a vector $x^u \in \{0,1\}^k$ where $x_j^u = 1$ if group G^j is allocated the item by $\mathcal{M}_u$ and $x_j^u = 0$ otherwise. Similarly, the lower allocation for group G^j is a vector $x^l \in \{0,1\}^{n_j}$ where $x_i^l = 1$ if bidder i^j is allocated the item and $x_i^l = 0$ otherwise.

The lower mechanism aggregates bids by calculating each group's *willingness to pay*. We define a group G^j's willingness to pay $\mathrm{WTP}^j = \mathrm{WTP}(b_1^j, \ldots, b_{n_j}^j)$ as the maximum amount the group can pay assuming that everyone who gets the item will pay the same amount (and no bidder pays more than its bid).[5] That is, the aggregation rule is given by

$$a(b_1^j, \ldots, b_{n_j}^j) = \mathrm{WTP}(b_1^j, \ldots, b_{n_j}^j) = \max_{i=1,\ldots,n_j} \{i b_i^j\}.$$

Let $t^j(p) = \max_{i=1,\ldots,n_j}\{i \mid i b_i^j \geq p\}$. In words, $t^j(p)$ is the largest number of bidders in group j that could be allocated the item and pay equally for it (without any bidder paying larger than its bid) *assuming the group is charged p by the auctioneer.* Note that, if $p \leq \mathrm{WTP}^j$, then $t^j(p)$ is well defined. If group j is allocated the item by the upper mechanism and charged p^j by the auctioneer, we define the lower allocation and cost sharing rules as follows for all $i = 1, \ldots, n_j$:

$$x_i^l(b_1^j, \ldots, b_{n_j}^j, S^j, p^j) = \begin{cases} 1 & \text{if } i \leq t^j(p^j) \\ 0 & \text{else} \end{cases}$$

$$c_i(b_1^j, \ldots, b_{n_j}^j, S^j, p^j) = \begin{cases} \frac{p^j}{t^j(p^j)} & \text{if } i \leq t^j(p^j) \\ 0 & \text{else} \end{cases}$$

Otherwise, if group j is not allocated the item, we simply have $x_i^l = 0$ and $c_i = 0$ for all $i \in [n_j]$.

The upper mechanism is a Vickrey auction. It takes all the group bids and allocates the item to the group with the highest bid and charges them the second highest group's

[5] We will see in Lemma 6 that this equal payment condition is necessary for incentive-compatibility (modulo some degenerate cases).

bid. Formalizing this, given group bids $b^1, \dots, b^k$ and letting j^* denote the index of the highest-bidding group:

$$x^u_j(b^1, \dots, b^k) = \begin{cases} 1 & \text{if } j = j^* \\ 0 & \text{else} \end{cases} \qquad p^j(b^1, \dots, b^k) = \begin{cases} \max_{j \neq j^*}\{b^j\} & \text{if } j = j^* \\ 0 & \text{else} \end{cases}$$

with ties broken arbitrarily.

Informally our mechanism works by having each group calculate their willingness to pay and bid that amount. Then the upper mechanism runs a standard second price auction. The winning group then splits the cost it is charged equally amongst the largest subset of its agents that it is able to. This subset of agents that are able to equally split the cost are exactly the agents the group gives access of the item to. Formally, the lower and upper mechanisms can be implemented together as follows. For the ease of exposition, any bidders whose allocations/payments aren't explicitly listed are implied to not be allocated any items and to have zero payment.

Algorithm 1 Single-Item Two-level Mechanism.

```
   Input: Bids $b^j = (b^j_1, \dots, b^j_{n_j})$ with $b^j_i \geq b^j_{i+1}$ for $j \in [k]$, $i \in [n_j - 1]$
1  for $j$=1,...,$k$ do
2      $\text{WTP}^j \leftarrow \max_{i \in [n_j]}\{i b^j_i\}$
3  $j^* \leftarrow \text{argmax}_{j \in [k]}\{\text{WTP}^j\}$
4  $p^{j^*} \leftarrow \max_{j \neq j^*}\{\text{WTP}^j\}$
5  $i^* \leftarrow \max_{i \in [n_{j^*}]}\{i | i b^{j^*}_i \geq p^{j^*}\}$
6  for $i = 1, \dots, i^*$ do
7      $x^{j^*}_i \leftarrow 1$
8      $p^{j^*}_i \leftarrow \frac{p^{j^*}}{i^*}$
9  return Allocation $\boldsymbol{x}$, payments $\boldsymbol{p}$
```

▶ **Theorem 2.** *Mechanism 1 is truthful, budget-balanced, and individually rational.*

Proof. We first show that the mechanism is truthful. Since this is a single parameter setting, it suffices to show that the allocation rule is monotone and that each winning bidder pays their critical bid.

We first show that the allocation rule is monotone. Assume bidder i^{j^*} is allocated the item by bidding $b^{j^*}_i$. Then if they increase their bid to $\tilde{b}^{j^*}_i \geq b^{j^*}_i$, we have that WTP^{j^*} weakly increases (that is, willingness-to-pay is weakly monotone in any coordinate). Thus, if group G^{j^*} wins when bidder i^{j^*} bids $b^{j^*}_i$, then group G^{j^*} will still win when bidder i^{j^*} bids $\tilde{b}^{j^*}_i$ with all other bidders' bids kept fixed. Furthermore, p^{j^*} would stay constant since the other bidders' bids stayed constant, and thus $b^{j^*}_i \geq \frac{p^{j^*}}{i^*}$ would imply $\tilde{b}^{j^*}_i \geq \frac{p^{j^*}}{i^*}$. Therefore, bidder i^{j^*} would still be allocated the item after increasing their bid, showing the allocation rule is monotone.

We now show that each winning bidder pays their critical bid, by showing that $\frac{p^{j^*}}{i^*}$ is the lowest that bidder i^{j^*} could drop their bid to while still winning (that is, bidder i^{j^*}'s payment is equal to its critical bid). Assume that bidder i^{j^*} drops their bid from $b^{j^*}_i$ to $\tilde{b}^{j^*}_i = \frac{p^{j^*}}{i^*}$. By the definition of i^* there are still at least $i^* - 1$ other bidders in G^{j^*} with bids at least $\frac{p^{j^*}}{i^*}$. Thus G^{j^*}'s WTP remains at least p^{j^*} implying that G^{j^*} still wins the upper auction. Furthermore, bidder i^{j*} would still have at least the i^*th highest bid in G^{j^*}. Thus we would still have $i^* \tilde{b}^{j^*}_i \geq p^{j^*}$ implying that bidder i^{j^*} would remain part of the winning subset of G^{j^*}.

If bidder i^{j^*} drops their bid to $b_i^{j^*}$ below $\frac{p^{j^*}}{i^*}$ then bidder i^{j^*} becomes at best the i^*th highest bidder in G^{j^*}. Even if G^{j^*} is still the winning group, G^{j^*} still has to pay the same payment p^{j^*}. This implies that $\max_{i=1,\ldots,n_{j^*}}\{i|ib_i^{j^*} \geq p^{j^*}\}$ weakly decreases by bidder i^{j^*} decreasing their bid. Thus if bidder i^{j^*} falls below the i^*th bid in G^j then they will no longer be part of the winning set. And, even if bidder i^{j^*} does become the i^*th highest bidder in G^j, then we would have $i^*\tilde{b}_i^{j^*} < i^*\frac{p^{j^*}}{i^*} = p^{j^*}$ implying that bidder i^{j^*} would no longer be part of the winning set. Thus bidder i^{j^*} would never be part of the winning set by dropping their bid below $\frac{p^{j^*}}{i^*}$. This shows that $p_i^{j^*} = \frac{p^{j^*}}{i^*}$ is bidder i^{j^*}'s critical bid for all winning bidders i^{j^*}.

Budget-balance is trivial for every losing group since every losing group is charged 0 by the auctioneer and doesn't have any of its members make payments. When the winning group is charged p^j, it chooses i^* bidders to pay $\frac{p^j}{i^*}$, showing that budget-balance holds there as well. Individual rationality follows since the only bidders that make payments are chosen such that (assuming truthful bids) $v_i^j \geq \frac{p^j}{i^*}$ and so $u_i^j = v_i^j - \frac{p^j}{i^*} \geq 0$. ◀

▶ **Theorem 3.** *Assuming truthful bidding, Mechanism 1 achieves an H_ℓ-approximation to the optimal social welfare, where ℓ is the maximum size of a group.*

Proof. We start with the following lemma giving a lower bound for the WTP of a group compared to the total value that group would get if every member was allocated the item.

▶ **Lemma 4.** *$WTP^j \geq \frac{W^j}{H_{n_j}}$ where $W^j = \sum_{i=1}^{n_j} v_i^j$ and H_i is the ith harmonic number.*

Proof. Note that $\mathrm{WTP}^j = \max_{i=1,\ldots,n_j}\{iv_i^j\}$. Thus we have,

$$\mathrm{WTP}^j \cdot H_{n_j} = \max_{i=1,\ldots,n_j}\{iv_i^j\}\sum_{i=1}^{n_j}\frac{1}{i} \geq \sum_{i=1}^{n_j}\frac{1}{i}iv_i^j = W^j. \quad ◀$$

Returning to the proof of Theorem 3. Note that if G^j is the group that wins the upper mechanism, then they obtain value at least WTP^j amongst their group members by allocating the item within their group. This is because if G^j wins, we have $\mathrm{WTP}^j \geq p^j$ and so $i^* = \max\{i|iv_i^j \geq p_j\} \geq \mathrm{argmax}_{i=1,\ldots,n_j}\{iv_i^j\}$. Thus, WTP^j is a lower bound of the value of the i^* bidders with the highest values in G^j being allocated the item.

Note that the optimal social welfare is achieved by the the group with the highest W^j to receive the item and for every member of that group to be allocated that item. Assume WLOG that this is G^1 and some group G^j wins the upper auction. Then we have that the mechanism achieves welfare at least WTP^j. If $G^j = G^1$ then we are done by Lemma 4; otherwise the fact that G^j won over G^1 in the upper auction implies $\mathrm{WTP}^j \geq \mathrm{WTP}^1 \geq \frac{W^1}{H_{n_1}}$. Since $H_{n_j} \leq H_\ell$, we have that the mechanism always achieves a H_ℓ fraction of the optimal social welfare. ◀

In general, the approximation factor achieved by Mechanism 1 is governed by the ratio of W^j and WTP^j. If this ratio is known to be smaller than H_ℓ – for example, because members of a common group tend to have similar valuations – then the guarantee of Theorem 3 improves accordingly.

3.2 Lower Bounds

We now show that the H_ℓ-approximation to welfare achieved by Mechanism 1 is in fact the best we can hope for from any truthful, budget-balanced mechanism satisfying equal treatment. We begin by bounding the maximum amount a group can be induced to pay compared to their true value for an item while maintaining incentive compatibility. Then we give a specific instance in which this occurs and show that any truthful budget balanced mechanism necessarily has to sometimes allocate items to lower-valued groups.

▶ **Theorem 5.** *There is no truthful, budget-balanced, individually rational two-level mechanism that satisfies equal treatment and guarantees more than an H_ℓ fraction of the optimal social welfare (where ℓ is the maximum group size).*

Proof. First, we can restrict attention to mechanisms that satisfy consumer sovereignty. If a mechanism doesn't satisfy consumer sovereignty, there exists an instance in which there is a bidder in some group that will never be allocated the item regardless of what they bid. Since the threshold price that a bidder needs to pay to be allocated the item is not a function of their bid for truthful mechanisms, this must hold regardless of the bidder's valuation. Letting that bidder's valuation tend to infinity then shows that worst-case welfare approximation of the mechanism is arbitrarily bad.

Next, note that because of the restriction that the aggregation function must be the identity function in the case in which a group only has 1 bidder (see Section 2), for the mechanism to be truthful, we must have that the upper mechanism is truthful with respect to group bids to be truthful with respect to individual bidders. Given this, we show the following result constraining the cost-sharing rule within the winning group.

▶ **Lemma 6.** *In single-item settings, except possibly on a set of valuation profiles with Lebesgue measure zero, a truthful and budget-balanced mechanism that satisfies equal treatment and consumer sovereignty must always charge all winning bidders the same payment.*

Proof. Note that winning bidders must all come from the same group. Then, because the upper mechanism is a truthful single-item auction, conditioned on G^j winning the item, the payment p^j that G^j has to make is independent of their bid b^j and hence $(b^j_1, \ldots, b^j_{n_j})$. Thus in a truthful, budget-balanced mechanism, G^j wins if and only if $\mathcal{M}_l$ is such that the payments its cost-sharing rule charges to winning bidders can cover the group's threshold payment $t^j(b^{-j})$. This makes $\mathcal{M}_l$ a truthful, budget-balanced cost sharing mechanism where $\mathcal{M}$ being truthful and budget-balanced implies that $\mathcal{M}_l$ will always be able to cover the cost it is asked to.

Next, we invoke a characterization result from [3, Theorem 3.4] that implies that, except possibly on a set of valuation profiles with Lebesgue measure zero, $\mathcal{M}_l$ must be the same lower mechanism as in Mechanism 1, meaning that it identifies the maximal subset of bidders $S \subset G^j$ such that each bidder $i^j \in S$ has $b^j_i \geq \frac{p^j}{|S|}$ and charges them all $\frac{p^j}{|S|}$. (Note that S is uniquely defined, as the union of two sets satisfying this property also satisfies that property.) In particular, every winning bidder makes the same payment. ◀

Returning to the proof of Theorem 5. Consider now an incentive-compatible two-level mechanism in which the lower mechanisms satisfy equal treatment. Recall that WTP^j as the largest amount that G^j can pay assuming all the winning bidders pay the same amount. By Lemma 6, the lower mechanisms must charge winning bidders a common amount (subject to individual rationality), so WTP^j is the maximum payment that could possibly be made by group G^j (except possibly on a measure-zero set of valuation profiles).

With this fact in hand, we consider two different instances, both with two groups. We assume that these instances do not fall into the measure-zero set of valuation profiles mentioned above; this assumption can always be enforced through arbitrarily small perturbations to the valuations, if needed.

Instance 1. G^1 has $n-1$ bidders with $v_i^1 = \frac{1}{i} - \delta$ for some $\delta > 0$ and G^2 has one bidder with $v_1^2 = 1$.

Instance 2. G^1 has one bidder with $v_1^1 = 1$ and G^2 has $n-1$ bidders with $v_i^2 = \frac{1}{i} - \delta$ for some $\delta > 0$.

We refer to Instance 1 by I_1 and Instance 2 by I_2. Note that the optimal welfare in both instances is H_{n-1} where all the bidders in G^1 win in I_1 and all the bidders in G^2 win in I_2. If G^2 wins in I_1 or G^1 wins in I_2 then the mechanism will only achieve a welfare of 1. Assume there is a truthful, budget-balanced two-level mechanism $\mathcal{M}$ that achieves an approximation factor better than H_{n-1}. Then, assuming that δ is sufficiently close to 0, $\mathcal{M}$ must have $\mathcal{M}_u$ choose G^1 in I_1 and G^2 in I_2.

Since $\mathcal{M}_u$ must be truthful with respect to group bids, it can be characterized by threshold payments $t^1(b^2)$ and $t^2(b^1)$ where G^1 winning implies $b^1 \geq t^1(b^2)$ and $b^2 \leq t^2(b^1)$ and G^2 winning implies the opposite. Since a is the identity function for groups with a single member, we have in I_1 that $b^2 = 1$ and in I_2 that $b^1 = 1$. We claim that G^1 winning in I_1 and G_2 winning in I_2 imply $\max\{t^1(1), t^2(1)\} \geq 1$.

Assume otherwise, and let $\max\{t^1(1), t^2(1)\} = y$ with $y < 1$. Then letting $\epsilon > 0$ such that $y + \epsilon < 1$, it follows that if $b^1 = y + \epsilon$ in I_1 and $b^2 = y + \epsilon$ in I_2 then we would still have G^1 win in I_1 and G^2 win in I_2. However this would imply $t^1(y+\epsilon) \geq 1 \implies t^1(1) < t^1(y+\epsilon)$ contradicting the monotonicity and hence truthfulness of $\mathcal{M}_u$.

Hence $\max\{t^1(1), t^2(1)\} \geq 1$ implies that either G^1 pays at least 1 in I_1 or G^2 pays at least 1 in I_2. However in I_1, we have $\text{WTP}(G^1) = 1 - \delta$ and in I_2, we have $\text{WTP}(G^2) = 1 - \delta$. Thus by the above lemma, there is no truthful budget-balanced mechanism satisfying equal treatment that will choose both G^1 to win in I_1 and G^2 to win in I_2 implying that no truthful, budget-balanced mechanism satisfying equal treatment can do better than a H_{n-1} approximation factor. ◀

This lower bound shows that the H_ℓ-approximation of Mechanism 1 is in fact tight and the best any incentive-compatible two-level mechanism could hope to do in this setting.

4 Multi-Unit Mechanisms

In this section, we move beyond the single-item setting and consider the case where the auctioneer is selling multiple items. Bidders can now have combinatorial valuations over the different items. In the case where bidders have additive valuations across the different items we show that our positive result for the single-item settings (Theorem 3) extends naturally.

The main result of this section shows that in general – and even with unit-demand valuations – we can't hope for any non-trivial approximation to optimal social welfare. In particular, we show that no truthful budget-balanced mechanism can do better than an n-approximation to social welfare in this setting, under very weak assumptions on the aggregation function.

4.1 Additive Valuations

For each of the items $l \in [m]$, each bidder i^j has some value $v_i^j(l) \in \mathbb{R}_+$. Then i^j's value for a set $S \subset [m]$ of items is $v_i^j(S) = \sum_{l \in S} v_i^j(l)$. The bidding language $\mathcal{B}$ consists of vectors $b_i^j \in \mathbb{R}_+^m$ where $b_i^j(l)$ specifies bidder i^j's value for item l. We will use $b^j(l)$ to refer to the vector of values all the bidders in G^j have for item l.

To extend our upper bound for single-item settings (Theorem 5) to the setting of additive valuations, it is sufficient to run an independent copy of Mechanism 1 for each of the items:

Algorithm 2 Additive Item Two-level Mechanism.

```
  Input: Bids $b^j = (b^j_1, ..., b^j_{n_j})$ for $j = 1, ..., k$;
         Single-Item Mechanism $\mathcal{M}$ (Mechanism 1)
1 for $l$=1,...,$m$ do
2     $(\tilde{x}, \tilde{p}) \leftarrow \mathcal{M}(b^1(l), ..., b^k(l))$
3     for $j = 1, ..., k$ do
4         for $i = 1, ..., n_j$ do
5             $p^j_i \leftarrow p^j_i + \tilde{p}^j_i$
6             $x^j_i(l) \leftarrow \tilde{x}^j_i$
7 return Allocation $\boldsymbol{x}$, payments $\boldsymbol{p}$
```

▶ **Theorem 7.** *Mechanism 2 is truthful, budget-balanced, individually rational and achieves a H_ℓ approximation to the optimal social welfare for bidders with additive valuations.*

Proof. Because a bidder's bid on one item has no affect on the allocation of or payments for any other item, truthfulness of Mechanism 2 follows straightforwardly from the truthfulness of Mechanism 1. Similarly, the budget-balance and individual rationality properties of Mechanism 2 follow easily from those of Mechanism 1 (and, in fact, hold on an item-by-item basis).

In the welfare-maximizing allocation, each item goes to the group that has the highest total value for that item and has all of its members allocated that item. From the analysis of Mechanism 1, we have that it allocates a given item to a set of bidders that have total value at least a $1/H_\ell$ fraction of the value any group has for that item. Because bidders have valuations that are additive across items, this implies that the welfare achieved by Mechanism 2 is within an H_ℓ factor of the optimal welfare. ◀

4.2 Unit-demand Valuations

We now consider bidders with unit-demand valuations. For each of the items $l \in [m]$, each bidder i^j has some value $v^j_i(l) \in \mathbb{R}_+$. Then i^j's value for a set $S \subset [m]$ of items is $v^j_i(S) = \max_{l \in S} v^j_i(l)$. The bidding language $\mathcal{B}$ consists of vectors $b^j_i \in \mathbb{R}^m_+$ where i^j specifies their bid for each item. In line with the auctioneer (of the upper mechanism) choosing a mechanism that is agnostic to group-specific idiosyncrasies, we require some kind of assumption that precludes a group from using its bid to signal information about its inner structure as opposed to a reasonable aggregation of its members' preferences. Many different such assumptions would be sufficient for our purposes; for concreteness, assume from now that the output of an aggregation function on a specific item must be bounded by a fixed but arbitrary function of the its inputs for that item. That is, there must exist some function $f : \mathbb{R}^n_+ \to \mathbb{R}_+$ such that for any item l and group G^j, $a(b^j_1, ..., b^j_{n_j})(l) \leq f(b^j_1(l), ..., b^j_{n_j}(l))$.

Within this setting we show that no truthful two-level mechanism can do better than a n approximation to the optimal welfare. The issue mechanisms in this setting face is not being able to distinguish whether a group bid representing a high value for many different items comes from that group having many bidders with disparate preferences or from a single unit demand bidder who is agnostic to which item they receive. In the interest of maintaining truthfulness, the mechanism is often forced to assume the group is composed of multiple disparate members and allocate to that group all the items they have high value for. However to remain truthful in the case where this valuation actually came from a single bidder with high value, the auctioneer can't charge more for a large bundle than it would charge for its individual components. In cases where different groups have similar preferences over the same items, this can cause only one of the groups being allocated the entire set of items hence harming the welfare in the case where these actually were just individual unit-demand bidders. We now proceed to making these ideas precise.

▶ **Theorem 8.** *No truthful two-level mechanism can achieve better than a n fraction of the optimal welfare.*

Proof. We can assume any mechanism that achieves at least a n-approximation to the optimal welfare satisfies consumer sovereignty. Since, if $\mathcal{M}$ does not satisfy consumer sovereignty, there must exist an instance where there is some item l that bidder i^j can't receive regardless of how high they bid. In this case let $v_i^j(l) \to \infty$. Then $\mathcal{M}$ will have an arbitrarily bad approximation factor to optimal welfare.

Throughout this proof we will use A_u^j to refer to the set of possible sets of items the upper mechanism can allocate to G^j. We start by showing that any truthful mechanism satisfying consumer sovereignty must have an upper mechanism that is able to allocate the set of all items to any group.

▶ **Lemma 9.** *Any truthful two-level mechanism that satisfies consumer sovereignty must satisfy $[m] \in A_u^j$ for all j.*

Proof. Let G^j have m unit-demand bidders, with bidder i^j's valuation function $v_i^j(l) = x$ if $i = l$ and otherwise $v_i^j(l) = 0$ where x is an arbitrary constant. Now fix the bids by the other groups. Since $\mathcal{M}_u$ must be truthful with respect to group bids, conditioned on $\mathcal{M}_u$ allocating a set $S \in A_u^j$ of items to G^j, G^j pays the same amount $p^j(S)$ regardless of its bid b^j. Using this along with consumer sovereignty, we claim there exists a value of x such that $\mathcal{M}_u$ must allocate $[m]$ to G^j to be truthful.

Assume that $\mathcal{M}_u$ does not allocate $[m]$ to G^j. It follows that there is some item l not allocated to G^j which further implies that l^j will have 0 utility in this allocation. However by consumer sovereignty, there exists a bid b_l^j that bidder l^j can make such that $\mathcal{M}_u$ will allocate some set S to G^j where $l \in S$ and furthermore l^j will be allocated l by G^j in $\mathcal{M}_l$. As we noted before the price $\mathcal{M}_u$ can charge to G^j for S is some fixed price $p^j(S)$, not a function of b^j. Thus consider the case where $x = \max_{S \in A_u^j}\{p^j(S)\} + 1$. By budget-balance, $\mathcal{M}_l$ can charge bidder l^j at most $p^j(S)$ given that G^j is charged $p^j(S)$ by $\mathcal{M}_u$. Thus bidder l^j can misreport their bid to be allocated l by $\mathcal{M}_l$, and as a result get a utility at least $\max_{S \in A_u^j}\{p^j(S)\} + 1 - p^j(S) > 0$. This shows a profitable non-truthful deviation for bidder l^j. This implies that when $x = \max_{S \in A_u^j}\{p^j(S)\} + 1$, G^j must be allocated $[m]$. Since we arbitrarily fixed the bids by the other groups, it follows that we must always have $[m] \in A_u^j$ for all j when $\mathcal{M}$ is truthful and satisfies consumer sovereignty. ◀

Lemma 9 shows that for a truthful mechanism $\mathcal{M}$ there must always be some bid b^j that G^j can make to be allocated all of $[m]$. We build upon this to show that in fact, for $\mathcal{M}$ to be truthful, for any arbitrary choice of bids b^{-j}, for every item l there exists some bid b^j where l is the highest value item in b^j and G^j is still allocated all of $[m]$. This shows that in such an instance, if G^j is actually comprised of a single bidder with valuation function b^j, they will still be allocated all of $[m]$ even though they are perfectly happy just receiving l. This will be used to cap the amount the mechanism can charge a group for the entirety of $[m]$ compared to any single item to remain truthful.

▶ **Lemma 10.** *For any truthful two-level mechanism $\mathcal{M}$ that achieves at least a n-approximation to welfare, for all items $l = 1, ..., m$, there always exists a b^j such that $x^u_j(b^1, ..., b^k) = [m]$ and $l = \text{argmax}_{l \in [m]} b^j(l)$.*

Proof. Following closely to the previous lemma, consider the following class of groups, parameterized by j. Let group $\tilde{G}^j$ be defined by having m bidders where each bidder prefers distinct items but the jth bidder in $\tilde{G}^j$ has a much larger value for their item compared to the other bidders in the group. Formally, for $j = 1, ..., m$ let group $\tilde{G}^j$ have m bidders where the unit demand valuation of bidder i^j is $v^j_i(l) = y$ if $i = j = l$, $v^j_i(l) = x$ if $i \neq j$ and $i = l$, and otherwise $v^j_i(l) = 0$ with y to later be defined as a function of x. Then fix the bids by other groups and let $x = \max_{S \in A^j_u}\{p^j(S)\} + 1$. Let the group bid for $\tilde{G}^j$ be $\tilde{b}^j$ and let $\boldsymbol{z}^i \in \mathbb{R}^m_+$ be vectors parameterized by $i = 1, ..., m$ where $\boldsymbol{z}^i(l) = x$ for $l = i$ and otherwise $\boldsymbol{z}^i(l) = 0$. By our assumption that the group bid for an item is bounded by some function of the individual bidders bids for an item, we have that there exists some function f such that $\tilde{b}^j(l) \leq f(\boldsymbol{z}^l)$ for $l \neq j$. Now let $g(x) = \max_{l \neq j}\{f(\boldsymbol{z}^l)\}$ and set $y = n^2(g(x) + x)$. Notably this means that the jth bidder in $\tilde{G}^j$ has much higher value for their item than the other bidders. We claim this implies that $\tilde{b}^j(j) \geq g(x)$.

Assume otherwise that $\tilde{b}^j(j) < g(x)$. Then define G' to be a group consisting of a single unit demand bidder. Let this bidder have a valuation of $v'(l) = ng(x)$ if $l = j$ and otherwise $v'(l) = 0$.

Now consider two instances both with two groups and m items.

Instance 1. G^1 is defined as $\tilde{G}^j$ above, and G^2 is defined as G'

Instance 2. G^1 is a single unit demand bidder with valuation $\tilde{b}^j$, and G^2 is defined as G'

We will refer to Instance 1 as I_1 and Instance 2 as I_2. In I_1 the optimal allocation is to give all the items to G^1 for a welfare of $n^2(g(x) + x) + (m - 1)x$, and in I_2 the optimal allocation is to give item j to G_2 and everything else to G^1 for a welfare of at least $ng(x)$.

Thus, if $\mathcal{M}_u$ allocates item j to G_1 then in I_2 the welfare is less than $g(x)$ implying an approximation factor strictly greater than $\frac{ng(x)}{g(x)} = n$. If $\mathcal{M}_u$ allocates item j to G_2 then in I_1 the welfare is at most $ng(x) + (m - 1)x$ giving an approximation factor of $\frac{n^2(g(x)+x)+(m-1)x}{ng(x)+(m-1)x} > n$, since here $n > m$. However, because the aggregation function is the identity for single bidder groups, I_1 and I_2 are indistinguishable to $\mathcal{M}_u$ and $\mathcal{M}_u$ being deterministic implies it must allocate item j to the same group in both instances. Thus if $\tilde{b}^j(j) < g(x)$ then $\mathcal{M}$ must have a worst case approximation factor worse than n.

Note that $\tilde{b}^j(j) \geq g(x)$ implies that $j \in \text{argmax}_{l \in [m]}(b^j(l))$. Furthermore from how we defined x and y, we must have that $\mathcal{M}_u$ allocates all of $[m]$ to $\tilde{G}^j$ following the proof from the previous lemma. Thus taking any fixed group bids b^{-j}. We can chose G^j to be defined as any of $\tilde{G}^i$ for $i = 1, .., m$ such that for all items $l = 1, .., m$ there exists a group bid b^j causing G^j to be allocated all of $[m]$ while having $l = \text{argmax}_{l' \in [m]}\{b^j(l)\}$. ◀

Note that $\mathcal{M}_u$ can't distinguish between the case when G^j consists of a single unit demand bidder and when G^j consists of multiple unit demand bidders. We show that this restrains the payments $\mathcal{M}_u$ can charge for allocating sets of items to groups by considering the case where every group is a single unit demand bidder. In characterizing the amount $\mathcal{M}_u$ can charge G^j for allocating G^j a set S of items, let the bids by all other groups be fixed. Then as before we denote the amount $\mathcal{M}_u$ charges G^j for S as $p^j(S)$. We start by showing that any truthful $\mathcal{M}_u$ can't charge less for a superset of another set of items. In the following lemmas we will refer to the unit demand valuations of the single bidder in G^j by v^j.

▶ **Lemma 11.** *For any truthful, two-level mechanism $\mathcal{M}$, let $S, T \in A_u^j$. Then $S \subset T$ implies that $p^j(S) \leq p^j(T)$.*

Proof. $p^j(S)$ and $p^j(T)$ must remain constant regardless of the inner structure of G^j, thus without loss of generality assume G^j consists of one bidder. Then for the sake of contradiction, assume that $p^j(S) > p^j(T)$. $S \in A_u^j$ implies that there exists some valuation v^j such that G^j is allocated S and charged $p^j(S)$. However $T \in A_u^j$ implies there is also an alternative bid $v^{j'}$ that G^j could make to be allocated T instead. Since v^j is a unit demand valuation, we have that $v^j(T) \geq v^j(S)$. Thus $p^j(S) > p^j(T)$ would imply that G^j profits by reporting $v^{j'}$ over v^j making $\mathcal{M}$ not truthful. ◀

We now show the more surprising statement that any truthful $\mathcal{M}_u$ can't charge more for $[m]$ than it would charge for any set S of items. This result stems from the fact that at times $\mathcal{M}_u$ has to allocate multiple items to a group that could be a unit demand bidder. Thus to maintain truthfulness, if G^j's favorite item is in S then G^j can't be charged more for $[m]$ than it would have been for S since it has the same value for both sets.

▶ **Lemma 12.** *For any truthful two-level mechanism $\mathcal{M}$ and set $S \in A_u^j$ where $S \neq \emptyset$, we have $p^j(S) \geq p^j([m])$*

Proof. As in the previous lemma, we can assume without loss of generality that G^j consists of 1 bidder. We have shown that $[m] \in A_u^j$ for any truthful mechanism. Thus there exists some valuation v^j such that $\mathcal{M}_u$ allocates $[m]$ to G^j. Let $l^* = \text{argmax}_{l\in[m]}\{v^j(l)\}$. It follows that for any set $S \in A_u^j$ such that $l^* \in S$ that $v^j(S) = v^j([m])$. $S \in A_u^j$ also implies there exists some bid $v^{j'}$ G^j could make to be allocated S. Thus if $p^j(S) < p^j([m])$ we would have that when G^j's value is v^j, G^j has a profitable deviation by bidding $v^{j'}$ instead making $\mathcal{M}$ not truthful. By lemma 10 we have that there exist valuations v^j such that $\mathcal{M}_u$ allocates $[m]$ to G^j where $l^* = l$ for any $l \in [m]$. Thus as long as $S \neq \emptyset$, we have that $p^j(S) \geq p^j([m])$. ◀

▶ **Corollary 13.** *In any truthful mechanism $\mathcal{M}$, for any 2 sets $S, T \in A_u^j$ we have $p^j(S) = p^j(T)\ \forall j = 1, ..., k$.*

This follows from the fact that for any set $S \in A_u^j, p^j(S) \leq p^j([m]), p^j(S) \geq p^j([m]) \implies p^j(S) = p^j([m])$. Finally we provide an instance that shows that no $\mathcal{M}_u$ that charges the same amount for all sets can do better than a n approximation of the optimal welfare.

▶ **Lemma 14.** *If $\mathcal{M}$ is truthful and $\mathcal{M}_u$ has $p^j(S) = p^j(T)\ \forall S, T \in A_u^j, j = 1, ..., k$ then $\mathcal{M}$ can not achieve a welfare approximation better than n.*

Proof. $\mathcal{M}_u$ being truthful with respect to group bids implies that the allocation A^j it gives to group G^j must satisfy $A^j = \text{argmax}_{A\in A_u^j}\{v^j(A) - p^j(A)\}$. We have that $p^j(A)$ is a constant for all $A \in A_u^j$, unless $A = \emptyset$ where $p^j(A) = 0$ by individual rationality, so let $p^j(A) = p^j$. However this implies that if $v^j(A) > p^j$ for any set A then $A^j = \text{argmax}_{A\in A_u^j}\{v^j(A)\}$. Let $A^{*j} \in \text{argmax}_{A\in A_u^j}\{v^j(A)\}$. Then if G^j is not allocated a set S where $v^j(S) = v^j(A^{*j})$, we have that $v^j(A^{*j}) \leq p^j$. This implies if $v(A) < v(A^{*j})$ then $v(A) < p^j$. Thus if $\mathcal{M}_u$ does not give G^j a set S where $v^j(S) = v^j(A^{*j})$, $\mathcal{M}_u$ must give G^j no items at all.

Given this consider the following instance with n identical groups where every group consists of a single unit demand bidder. There are n total items with each bidder in each group having an identical valuation vector of $v_1^j = (1+\epsilon, 1, ..., 1, 1)$. Call the first item that every bidder has value $1+\epsilon$ for l_1. Here we have that $v(S) = v(A^{*j})$ if and only if $l_1 \in S$. However note that $\mathcal{M}_u$ can only allocate l_1 to one of the groups. Let this group be $G^{j'}$. Then by the observation above we have that any groups G^j where $j \neq j'$ must not be allocated any items at all. Thus the maximum welfare any truthful $\mathcal{M}_u$ can achieve in this scenario is to allocate every item to $G^{j'}$ for a welfare of $1 + \epsilon$. However, since every group is a single unit demand bidder, the optimal allocation is to give every group a unique item for a welfare of $n + \epsilon$. Thus as $\epsilon \to 0$ we get that no $\mathcal{M}$ where $\mathcal{M}_u$ charges equal prices for every set of goods can achieve better than a n approximation of the optimal welfare. ◀

Thus since every truthful upper mechanism must charge the same prices for every subset of items in its range, we have that no truthful mechanism can achieve an approximation factor better than n. ◀

5 Conclusion

Our work can be extended in a number of ways; we list some of these below. Some of these directions strive to capture additional practical scenarios; others aim to enrich the model in hopes of bypassing the lower bounds proved in this paper.

Fractional allocations

In the present work, bidders either have full or no access to an item. An alternative model would allow bidders to be allocated a fraction of an item. One way to do this is to depart from the public excludable goods setting and to instead split an item fractionally among the bidders in the winning group (like time-sharing a performance venue). One practical example of this in the context of DAOs is distributing voting rights: once a DAO wins an item in an auction it can then allocate more voting power over how the item is used to bidders who bid higher and correspondingly charge them more as well. It would be interesting to obtain positive results in this model, possibly under relaxed notions of budget-balance or incentive-compatibility.

An alternative way to introduce fractional allocations is to remain in the public excludable goods framework of allowing the lower mechanism to choose any allocation it likes amongst its members, but giving it the additional power of allocating fractional items (i.e., "partial access" to the item) to players at a lower cost . It is an open question whether, in the single-item setting, this extra power can be used to design a two-level mechanism that is truthful, budget-balanced, individually rational, and achieves a better-than-H_ℓ-approximation of the optimal social welfare.

Randomized mechanisms

One natural extension of this work is to consider randomized mechanisms. We note that the lower bounds for truthful budget-balanced cost-sharing mechanisms in [3] (for the social cost minimization objective) do not immediately carry over to our model (and the social welfare maximization objective). For example, a randomized mechanism that outputs an optimal solution (with high social welfare and social cost near 0) with 50% probability and a terrible solution (with zero social welfare and high social cost) with 50% probability would achieve a good approximation of the social welfare objective but not the social cost objective. It is an open question whether, in the single-item setting, there is a randomized two-level mechanism that is truthful, budget-balanced, individually rational, and achieves an $o(H_\ell)$-approximation of the optimal social welfare.

Bayes-Nash equilibria

If the compromises required by truthfulness are too much to bear, non-truthful mechanisms can be considered instead. Could there be a simple two-level mechanism such that, under suitable assumptions about the distribution over bidders' valuations, guarnatees near-optimal social welfare at Bayes-Nash equilibrium?

Communication vs. welfare tradeoffs

The lower bounds in this paper show that the compression required from aggregation functions (from a set of member valuations to a summary group valuation) leads to groups incompletely representing the preferences of their members, which in turn results in a loss of social welfare. At the other extreme, if groups can simply pass on the collection of their members' valuations to the top-level mechanism (with no aggregation), optimal social welfare can be achieved by a truthful two-level mechanism that essentially implements the VCG mechanism. It would be interesting to explore the seemingly inherent tradeoff between the bound on the output complexity of an aggregation functions and the best-possible social welfare approximation achievable by a truthful two-level mechanism.

Disclosures

The authors are researchers at a16z Crypto, a venture capital firm with investments in a broad range of web3 projects (for general a16z disclosures, see `https://www.a16z.com/disclosures/`).

A Truthful Mechanisms Don't Compose

We give here an explicit example showing that a two-level mechanism $\mathcal{M}$ in which $\mathcal{M}_u$ implements a truthful auction mechanism and $\mathcal{M}_l$ implements a truthful cost sharing mechanism doesn't necessarily imply that $\mathcal{M}$ itself is truthful two-level mechanism. To see this, we will consider a simple instance with unit-demand bidders and consider the two-level mechanism $\mathcal{M}$ where $\mathcal{M}_u$ implements VCG and $\mathcal{M}_l$ implements the "maximum equal-split" cost shares used in Mechanism 1 (see Section 3), which are arguably the most canonical choices for the upper and lower mechanisms. We show that, no matter what aggregation function is used, this two-level mechanism cannot be truthful.

Consider the following instance with 2 groups G^1 and G^2 competing for 2 items r and s where G^1 has 2 agents and G^2 has 1 agent. Let the agents' valuations be $v_1^1 = (10+2\epsilon, 0), v_2^1 = (5-\epsilon, \epsilon), v_1^2 = (10, \epsilon)$ with the first indices and second indices referring to agents' values for r and s respectively. Then let $\mathcal{M}$ be a two-level mechanism where $\mathcal{M}_u$ implements VCG and $\mathcal{M}_l$ implements maximum equal-split cost shares. Then since we would have $b^2 = (10, \epsilon)$, regardless of the aggregation function used to create b^1, $\mathcal{M}_u$ will always give at least one of the items to G^2 as the welfare maximizing outcome according to group bids. Then by the VCG payment rule, in the case that G^1 wins r $p^1 = 10 - \epsilon$, and in the case that G^2 wins s, $p^1 = 0$.

With our choice of cost-sharing, all winning agents in a group pay the same amount. Thus, since agent 2^1's value for r is only $5 - \epsilon$, $\mathcal{M}_l$ will only allocate r to agent 2^1 if $p^1 \leq 10 - 2\epsilon$. Thus $p^1 > 10 - 2\epsilon$ implies that agent 2^1 is not allocated r and has 0 utility in this case. However, by consumer sovereignty, there exists some bid agent 2^1 can make that causes G^1 to be allocated item s instead. Then $p^1 = 0$ so both agents in G^1 will be allocated s making

$u_2^1 = \epsilon > 0$. Thus no truthful mechanism can give G^1 only item r. However, if $\mathcal{M}_u$ only gives G^1 item s, then agent 1^1 is guaranteed to have 0 utility. Thus again by consumer sovereignty, there exists some bid b_1^1 agent 1^1 can make such that G^1 is allocated r instead. In this case agent 1^1 has to pay at most $10 - \epsilon$ by budget balance making $u_1^1 \geq 10 + 2\epsilon - 10 + \epsilon = 3\epsilon > 0$. Thus regardless of whether the aggregation function causes $\mathcal{M}_u$ running VCG to give G^1 either r or s, there always exists a profitable non-truthful deviation by either agent 1^1 or agent 2^1 showing that $\mathcal{M}$ is never truthful in this setting.

References

1 Gagan Aggarwal, Kshipra Bhawalkar, Guru Guruganesh, and Andrés Perlroth. Maximizing revenue in the presence of intermediaries. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 1:1–1:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

2 Moshe Babaioff, Moran Feldman, and Moshe Tennenholtz. Mechanism design with strategic mediators. *ACM Transactions on Economics and Computation (TEAC)*, 4(2):1–48, 2016.

3 Shahar Dobzinski, Aranyak Mehta, Tim Roughgarden, and Mukund Sundararajan. Is shapley cost sharing optimal? *Games Econ. Behav.*, 108:130–138, 2018.

4 Shahar Dobzinski and Jan Vondrák. The computational complexity of truthfulness in combinatorial auctions. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 405–422, 2012.

5 Joan Feigenbaum, Christos Papadimitriou, and Scott Shenker. Sharing the cost of muliticast transmissions (preliminary version). In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 218–227, 2000.

6 Jon Feldman, Vahab Mirrokni, S. Muthukrishnan, and Mallesh M. Pai. Auctions with intermediaries: Extended abstract. In *Proceedings of the 11th ACM Conference on Electronic Commerce*, EC '10, pages 23–32, New York, NY, USA, 2010. Association for Computing Machinery.

7 Daniel A Graham and Robert C Marshall. Collusive bidder behavior at single-object second-price and english auctions. *Journal of Political economy*, 95(6):1217–1239, 1987.

8 Daniel Lehmann, Liadan Ita Oćallaghan, and Yoav Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM (JACM)*, 49(5):577–602, 2002.

9 Kevin Leyton-Brown, Yoav Shoham, and Moshe Tennenholtz. Bidding clubs: institutionalized collusion in auctions. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pages 253–259, 2000.

10 Kevin Leyton-Brown, Yoav Shoham, and Moshe Tennenholtz. Bidding clubs in first-price auctions. In *AAAI/IAAI*, pages 373–378, 2002.

11 Robert C Marshall and Leslie M Marx. Bidder collusion. *Journal of Economic Theory*, 133(1):374–402, 2007.

12 R Preston McAfee and John McMillan. Bidding rings. *The American Economic Review*, pages 579–599, 1992.

13 Hervé Moulin and Scott Shenker. Strategyproof sharing of submodular costs: budget balance versus efficiency. *Economic Theory*, pages 511–533, 2001.

14 Noam Nisan and Amir Ronen. Algorithmic mechanism design. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 129–140, 1999.

15 Christos Papadimitriou, Michael Schapira, and Yaron Singer. On the hardness of being truthful. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 250–259. IEEE, 2008.

16 Shiran Rachmilevitch. Auctions with multi-member bidders. working paper, 2022.

17 Kevin Roberts. The characterization of implementable choice rules. *Aggregation and revelation of preferences*, 12(2):321–348, 1979.

18 Tim Roughgarden and Mukund Sundararajan. Quantifying inefficiency in cost-sharing mechanisms. *J. ACM*, 56(4):23:1–23:33, 2009.
19 Tim Roughgarden and Inbal Talgam-Cohen. Approximately optimal mechanism design. *Annual Review of Economics*, 11:355–381, 2019.

Fast and Furious Withdrawals from Optimistic Rollups

Mahsa Moosavi
Concordia University, Montreal, Canada
OffchainLabs, Princeton, NJ, USA

Mehdi Salehi
OffchainLabs, Princeton, NJ, USA

Daniel Goldman
OffchainLabs, Princeton, NJ, USA

Jeremy Clark
Concordia University, Montreal, Canada

Abstract

Optimistic rollups are in wide use today as an opt-in scalability layer for blockchains like Ethereum. In such systems, Ethereum is referred to as L1 (Layer 1) and the rollup provides an environment called L2, which reduces fees and latency but cannot instantly and trustlessly interact with L1. One practical issue for optimistic rollups is that trustless transfers of tokens and ETH, as well as general messaging, from L2 to L1 is not finalized on L1 until the passing of a dispute period (aka withdrawal window) which is currently 7 days in the two leading optimistic rollups: Arbitrum and Optimism. In this paper, we explore methods for sidestepping the dispute period when withdrawing ETH from L2 (called an exit), even in the case when it is not possible to directly validate L2. We fork the most-used rollup, Arbitrum Nitro, to enable exits to be traded on L1 before they are finalized. We also study the combination of tradeable exits and prediction markets to enable insurance for withdrawals that do not finalize. As a result, anyone (including contracts) on L1 can safely accept withdrawn tokens while the dispute period is open despite having no knowledge of what is happening on L2. Our scheme also allows users to opt-into a fast withdrawal at any time. All fees are set by open market operations.

2012 ACM Subject Classification Security and privacy; Security and privacy → Cryptography

Keywords and phrases Ethereum, layer 2, rollups, bridges, prediction markets

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.22

Supplementary Material *Software (Source Code)*: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals` archived at `swh:1:dir:5e160d45808e5387baa1e65eff88f6eed0e37c97`

Funding *Jeremy Clark*: acknowledges support for this research project from (i) the National Sciences and Engineering Research Council (NSERC), Raymond Chabot Grant Thornton, and Catallaxy Industrial Research Chair in Blockchain Technologies (IRCPJ/545498-2018), (ii) the Autorité des Marchés Financiers, and (iii) a NSERC Discovery Grant (RGPIN/04019-2021).

Acknowledgements This paper includes useful comments from the reviewers, discussions with Edward W. Felten and Rachel Bousfield, and feedback from presentations at Devcon 6 and a16z crypto research.

1 Introductory Remarks

Ethereum-compatible blockchain environments, called Layer 2s (or L2s) [5], have demonstrated an ability to reduce transaction fees by 99–99.9% while preserving the strong guarantees of integrity and availability in the underlying Layer 1 (or L1) blockchain. The subject of this paper concerns one subcategory of L2 technology called an optimistic rollup. The website *L2 Beat* attempts to capitalize all tokens of known value across the top 25 L2 projects. It finds that the top two L2s are both optimistic rollups, *Arbitrum* and *Optimism*, which respectively account for 50% and 30% of all L2 value – $4B USD at the time of writing. [1]

We will describe the working details of optimistic rollups later in this paper but here are the main takeaways: currently, rollups are faster and cheaper than Ethereum itself. However, each L2 is essentially an isolated environment that cannot instantly and trustlessly interact with accounts and contracts that are running on either L1 or other L2s. An optimistic rollup project will typically provide a smart contract, called a validating bridge [9], that can trustlessly move ETH (and other tokens and even arbitrary messages) between L1 and its own L2. It implements a transfer by locking the ETH in an L1 contract and minting the equivalent ETH on L2 and assigning it to the user's L2 address. More precisely, L2 ETH is a transferrable claim for L1 ETH from the L1 bridge at the request of the current owner of the L2 claim. Later when the user requests a withdrawal, the ETH will be destroyed on L2 and released by the bridge back onto L1 according to whom its new owner is on L2 at the time of the request. This requires the rollup to convince the L1 bridge contract of whom the current owner of withdrawn ETH is on L2. We provide details later but this process takes time: the bridge has to wait for a period of time called the dispute window. The current default is 7 days in *Arbitrum* and *Optimism*, however the filing of new disputes can extend the window. The bottom line is that users have to wait at least 7 days to draw down ETH from an optimistic rollup.

Contributions

In this paper, we compare several methods – atomic swaps and tradeable exits – for working around this limitation. While we argue workarounds cannot be done generally (*e.g.,* for NFTs, function outputs, or arbitrary messages), some circumstances allow it: namely, when the withdrawn token is liquid, fungible, and available on L1 and the withdrawer is willing to pay a fee to speed up the withdrawal. While these techniques work easily between human participants that have off-chain knowledge, such as the valid state of the L2, it is harder to make them compatible with L1 smart contracts that have no ability to validate the state of L2. We propose a solution using tradeable exits and prediction markets to enable an L1 smart contract to safely accept withdrawn tokens before the dispute period is over. We fork the current version, *Nitro*, of the most used optimistic rollup, *Arbitrum*, maintained as open source software[2] by *Offchain Labs*. *Arbitrum* is a commercial product with academic origins [8]. We implement our solution and provide measurements. We also provide an analysis of how to price exits and prediction market shares.

2 Background

While we describe optimistic rollups as generally as possible, some details and terms are specific to *Arbitrum*.

2.1 Inbox

Rollups have emerged as a workable approach to reduce fees and latency for Ethereum-based decentralized applications. In a rollup, transactions to be executed on L2 are recorded in an L1 smart contract called the inbox. Depending on the system, users might submit to

[1] L2 Beat: `https://l2beat.com/scaling/tvl/`, accessed Oct. 2022.
[2] GitHub: Nitro `https://github.com/OffchainLabs/nitro`

the inbox directly, or they might submit to an offchain service, called a sequencer, that will batch together transactions from many users and pay the L1 fees for posting them into the inbox. Transactions recorded in the inbox (as `calldata`) are not executed on Ethereum, instead, they are executed in a separate environment off the Ethereum chain, called L2. This external environment is designed to reduce fees, increase throughput, and decrease latency.

2.2 Outbox

Occasionally (*e.g.,* every 30–60 minutes), validators on L2 will produce a checkpoint of the state of all contracts and accounts in the complete L2 according to the latest transactions and will place this asserted state (called an RBlock) in a contract on L1 called the outbox. Note that anyone with a view of L1 can validate that the sequence of transactions recorded in the inbox produces the asserted RBlock in the outbox. This includes Ethereum itself, but asking it to validate this be equivalent to running the transactions on Ethereum. The key breakthrough is that the assertion will be posted with *evidence* that the RBlock is correct so Ethereum does not have to check completely.

2.3 Optimistic vs. zk-rollups

In practice, two main types of evidence are used. In zk-rollups,[3] a succinct computational argument that the assertion is correct is posted and can be checked by Ethereum for far less cost than running all of the transactions. However the proof is expensive to produce. In optimistic rollups, the assertions are backed by a large amount of cryptocurrency acting as a fidelity bond. The correctness of an RBlock can be challenged by anyone on Ethereum and Ethereum itself can decide between two (or more) RBlocks for far less cost than running all of the transactions (by having the challengers isolate the exact point in the execution trace where the RBlocks differ). It will then reallocate the fidelity bonds to whoever made the correct RBlock. If an RBlock is undisputed for a window of time (*e.g.,* 7 days), it is considered final.

2.4 Bridge

A final piece of the L2 infrastructure is a bridge, which can move ETH, tokens, NFTs, and even arbitrary messages, between L1 and L2. Our fast withdrawals is limited to ETH and fungible tokens. If Alice has ETH on Ethereum, she can submit her ETH to a bridge smart contract on Ethereum which will lock the ETH inside of it, while generating the same amount of ETH in Alice's account inside the L2 environment. The bridge does not need to be trusted because every bridge operation is already fully determined by the contents of the inbox. Say that Alice transfers this ETH to Bob's address on L2. Bob is now entitled to draw down the ETH from L2 to L1 by submitting a withdrawal request using the same process as any other L2 transaction – *i.e.,* placing the transaction in the inbox on L1, having it executed on L2, and seeing it finalized in an RBlock on L1. Optimistically, the RBlock is undisputed for 7 days and is finalized. Bob can now ask the bridge on L1 to release the ETH to his address by demonstrating his withdrawal (called an exit) is included in the finalized RBlock (*e.g.,* with a Merkle-proof).

[3] zk stands for zero-knowledge, a slight misnomer: succinct arguments of knowledge that only need to be complete and sound, not zero-knowledge, are used [10].

2.5 Related Work

Arbitrum is first described at *USENIX Security* [8]. Gudgeon *et al.* provide a systemization of knowledge (SoK) of Layer 2 technology (that largely predates rollups) [5]. McCorry *et al.* provide an SoK that covers rollups and validating bridges [9], while Thibault *et al.* provide a survey specifically about rollups [13]. Some papers implement research solutions on Arbitrum for improved performance: decentralized order books [11] and secure multiparty computation [2]. The idea of tradeable exits predates our work but is hard to pinpoint a source (our contribution is implementation and adding hedges). Further academic work on optimistic rollups and bridges is nascent – we anticipate it will become an important research area.

Other related topics are atomic swaps and prediction markets. Too many papers propose atomic swap protocols to list here but see Zamyatin *et al.* for an SoK of the area (and a new theoretical result) [14]. Decentralized prediction markets proposals predate Ethereum and include Clark *et al.* [1] and Truthcoin [12]. Early Ethereum projects *Augur* and *Gnosis* began as prediction markets.

3 Proposed Solution

For simplicity, we will describe a fast exit system for withdrawing ETH from L2, however it works for any L1 native fungible token (*e.g.,* ERC20) that is available for exchange on L1. We discuss challenges of fast exits for non-liquid/non-fungible tokens in Section 6.4. Consider an amount of 100 ETH. When this amount is in the user's account on L1, we use the notation 100 ETH_{L1}. When it is in the bridge on L1 and in the user's account on L2, we denote it 100 ETH_{L2}. When the ETH has been withdrawn on L2 and the withdrawal has been asserted in the L1 outbox, but the dispute window is still open, we refer to it as 100 ETH_{XX}. Other transitionary states are possible but not needed for our purposes.

3.1 Design Landscape

In Table 1, we compare our solution to alternatives in industry and the blockchain (academic and grey) literature that could be used for fast withdrawals.

3.1.1 Properties

We are interested in solutions that do not require a trusted third party. If trust is acceptable, a centralized exchange that has custody of its users funds is a fast and user-friendly solution. We consider anything faster than the 7-day dispute period as "fast" but take measurements of solutions that can settle within a fully confirmed "L1 transaction" (*e.g.,* minutes) and within a unconfirmed L2 RBlock (*e.g.,* hours). This assumes that all counterparties perform instantly upon request. Settlement is from the perspective of the withdrawer, Alice, only and does not necessarily mean other counterparties will complete within the same timeframe. For example, in many solutions, Alice will have her withdrawn ETH quickly at the expense of a counterparty waiting out the dispute period.

Some solutions require one party to act, followed by an action of the counterparty in a follow-up transaction. This creates the risk that the counterparty aborts the protocol before taking their action. Since it is unknown if the counterparty will act or not, these protocols establish a window of time for the counterparty to act and if the window passes without action, the initial party has to begin the protocol again with a new counterparty. The protocols ensure that funds are never at risk of being lost, stolen, or locked up forever, however the

Table 1 Comparing alternatives for fast withdrawals from optimistic rollups for liquid and fungible tokens where • satisfies the property fully, ◦ partially satisfies the property, and no dot means the property is not satisfied. ⊥ was not measured. For our work, ∼ means we propose how to fully achieve the property but do not by default (see caveats in Section 6.1).

Type	*Example*	No trusted third party	Within an L1 transaction	Within an L2 rollup	No griefing	No free option	Opt-in anytime	Crosschain or L2-to-L2	L1 gasUsed	L2 gasUsed	Other Fees
Normal Exit (baseline)	Arbitrum	•			•	•			200K	80K	–
Centralized	Binance		•	•	•	•		•	400K	21K	Operator
HTLC Swaps	Celer	•	◦	•				•	625K	92K	
Conditional Transfers	StarkEx	•	•	•					⊥	⊥	Operator
Bridge Tokens	Hop	◦	•	•		•		•	1.8M	300K	Operator
Tradeable Exits	This Work	•	∼	•	•	•	•		200K	80K	Discount
Hedged Tradeable Exits	This Work	•	∼	•	•	•	•		265K	80K	FAIL_{PM}

protocols admit two smaller issues. The first issue is that a malicious counterparty could accept to participate with no intention of completing the protocol just to "grief" the party taking the action – wasting their time and possibly gas fees for setting up and tearing down the conditions of the trade. The second issue is that a strategic counterparty can accept to participate and then selectively choose to complete or abort, as well as timing exactly when they choose to complete (within the window), based on price movements or other market information. This is called (somewhat cryptically) a "free option;" finance people might recognize it as akin to being given an American call option for free.

A solution is "opt-in anytime" if the user can withdraw normally and then (say upon realizing for the first time that there is a 7 day dispute window) decide to speed up their transaction. While it is not a design goal of our paper, many of these solutions are generic cross-chain transactions (including L2-to-L2 swaps). A drawback of our solution is that it is narrowly scoped to L2-to-L1 withdrawals on rollups. Therefore our solution is not intended as a complete replacement of atomic swaps or the other solutions in Table 1. It is designed to be best-in-class only for slow rollup withdraws.

Finally we estimate the costs involved for the seller of ETH_{L2}. For some protocols, the gas cost of the buyer might differ from the seller depending if its actions are symmetric or not – we comment on this but did not find it interesting enough to put in the table. The more interesting aspect is that many alternatives do require a third party to be involved (we generically call them "operators") and they must be compensated for their actions. In some alternatives, the operators might be not be inherently necessary (*e.g.,* an HTLC swap) but are used in practice (*e.g.,* Celer) to ease friction (*e.g.,* users finding other users to swap with): in this case, we are charitable and do not mark the fee. So the fees are for things fundamental to how the alternative works. We expand more within the discussion of each alternative below.

3.1.2 Alternatives

Centralized

Consider Alice who has 100 ETH_{L2} and wants 100 ETH_{L1} for it. A centralized exchange (*e.g., Coinbase*, *Binance*) can open a market for $\text{ETH}_{\mathsf{L2}}/\text{ETH}_{\mathsf{L1}}$. Alternatively, a bridge might rely on an established set of trustees to relay L2 actions to L1. This is called proof of authority; it is distributed but not decentralized (*i.e.,* not an *open* set of participants). The gas costs consists of Alice transferring her ETH_{L2} onto the exchange (withdraw to L1 is paid for by the exchange). An exchange will not be profitable if it offers this for free, therefore it captures a operator fee for the service.

Hash Time Locked Contracts (HTLCs)

Assume Bob has 100 ETH_{L1} and is willing to swap with Alice. An atomic swap binds together (i) an L2 transaction moving 100 ETH_{L2} from Alice to Bob and (ii) an L1 transaction moving 100 ETH_{L1} from Bob to Alice. Either both execute or both fail. HTLC is a blockchain-friendly atomic swap protocol. Its main drawback is that it also has a time window where Alice (assuming she is the first mover in the protocol) must wait on Bob, who might abort causing Alice's ETH_{L2} to be locked up while waiting (called the griefing problem), or might watch price movements before deciding to act (called free option problem). Bob needs to monitor both chains so he cannot be an autonomous smart contract. HTLCs can work generically between any two chains capable of hash- and time-locking transaction outputs; this includes between two L2s.

The transaction (containing a hashlock and timeout) is slightly more complicated than a standard ETH transfer, requiring smart contract logic on both layers. The measurement based on Celer is not a pure HTLC and uses operators as well for liquidity and staking, but we omit these fees from the table because theoretically Alice and Bob could find each other and perform a pure HTLC with no added infrastructure.

Conditional Transfers

The intuition behind a conditional transfer (CT) is that L1-to-L2 messaging (or bridging) is fast even if L2-to-L1 messaging is slow. CT exploits this to build an HTLC-esque swap specifically for withdrawing from rollups (while HTLCs are designed generically for cross-chain swaps). Alice beings by registering her intent to trade 100 ETH_{L2} for 100 ETH_{L1} in a special registry contract on L1, and she locks (*e.g.,* for an hour) 100 ETH_{L2} in escrow on L2. If Bob agrees to the swap, Alice provides him (off-chain) with a signed transaction (called the conditional transfer) that transfers the escrowed 100 ETH_{L2} to Bob, conditioned on Alice having receiving 100 ETH_{L1} in the registry contract on L1. After Bob transfers the ETH_{L1} on L1, this fact can be bridged to the L2 escrow contract (with customization of the rollup's inbox) quickly (recall that L1-to-L2 messaging is fast). The L2 escrow contract will flag that the L1 transaction has paid by Bob, and Bob can broadcast his signed (by Alice) L2 transaction to recover 100 ETH_{L2} from escrow (if Bob broadcasts it before the flag is set, it simply reverts).

In terms of existing implementations, we could not adequately isolate the conditional transfer component from the rest of the bridge to measure gas costs (denoted in the table using a $\perp$ symbol) however it should be slight more expensive than an HTLC as the logic of the transaction is more complex.

Also note that Bob must be a validator on L2 to confirm that the state of the escrow and conditional transfer on L2 will result in him being paid – this is where the speedup really comes from, if he waits for L1 to finalize this, then the transfer happens after the dispute period and it is no different than a normal exit. Consequently, Bob cannot be an autonomous L1 smart contract unable to validate L2 state until it is finalized on L1 (which is the design goal of our alternative: hedged tradeable exits).

Bridge Token

A bridge token is not a novel technical innovation but it is a practical market design for supplying bridges with liquidity. Bridges between L1 and L2 can technically be implemented by anyone. It is natural for the inbox/outbox provider to provide a bridge but it is not strictly necessary.

Assume a third party creates a contract on L1 that accepts ETH_{L1} and releases a transferable claim for ETH_{L1}; it creates the same contract on L2. Assume enough of these claims come into circulation that a liquid market for them emerges on both layers. To move ETH_{L2} to ETH_{L1}, Alice starts by trading her ETH_{L2} for a claim to the same amount on L2. She then asks the L2 contract to transfer the claim which it does by burning them and firing an event. An authorized party, called a bonder, notices the event on L2, goes to the L1 contract and mints the same number of claims on L1 for ETH_{L1} and transfers them to Alice's address. Technically the L1 contract is insolvent as more claims exist than actual ETH_{L1} in the contract, but the L2 contract is oversolvent by the same amount. The contracts can be rebalanced (1) through movements in the opposite direction; (2) through a bulk withdrawal after the normal 7-day dispute period; or (3) by incentivize bonders to purposefully rebalance the contracts by burning claims on L1 and minting on L2. To prevent the bonder from maliciously minting tokens on L1 that were not burned on L2, it must post a fidelity bond of equal or greater value. (Alternatively, the bonder can be a trusted party which makes it the same in analysis as a centralized exchange). After the 7-day dispute period, the L1 contract can verify the bonder's actions are consistent with the burns on L2 and release its fidelity bond.

Note that when you collapse this functionality, it is equivalent to the bonder buying ETH_{XX} from Alice for ETH_{L1} and receiving their ETH_{L1} back 7 days later. The extra infrastructure is necessary because today native bridges do not support transferring ETH_{XX}. As in atomic swaps, the bonder can fail to act (griefing) which is worst in this case if Alice cannot "unburn" her tokens, but there is no free option because Bob is a relay and not a recipient of the tokens. The gas fee measurement is based on Hop and standard token transfers on L1 and L2. The main cost of bridge tokens is paying the bonder (called an operator in the table) who are providing a for-profit service.

3.2 Tradeable Exits

Alice wants to withdraw 100 ETH_{L2}. Unlike the other solutions, Bob takes the risk that the exit never finalized and therefore will offer less than 100 ETH_{L1} (say 99.95 ETH_{L1}) for it (this is denoted "discount" in Table 1). Assume Bob has 99.95 ETH_{L1} that will not use until after the dispute window. Bob also runs an L2 validator so he is assured that if Alice withdraws, it is valid and will eventually finalize. With a tradeable exit, the outbox allows Alice to change the recipient of her withdraw from herself to Bob. Thus Alice swaps her pending exit of 100 ETH_{L1} (which we call 100 ETH_{XX}) for Bob's 99.95 ETH_{L1} on L1 (note we discuss the actual difference in price in Section 5). Since ETH_{L1} and ETH_{XX} are both on

L1, Alice can place an ask price for her ETH_{XX} and the first trader willing to swap can do so atomicly, with no ability to grief or capitalize on a free option. After 7 days, Bob can ask the bridge to transfer the ETH_{L1} to his address, and the bridge checks the outbox to validate that Bob's address is the current owner of the exit.

In our forked bridge, Alice can transfer any of her exits that are in an RBlock (*i.e.,* an asserted L2 state update registered in the outbox). Technically, Bob can check the validity of the withdrawal as soon as it is in the inbox, and not wait 30-60 minutes for an RBlock. However for implementation reasons, it is easier to track an exit based on its place (*i.e.,* Merkle path) in an RBlock, rather than its place in the inbox. When we say a withdrawal is "fast," we mean 30-60 minutes (*i.e.,* one L2 rollup).

Tradeable exits can be approximated by a third party L1 contract that does not modify the rollup. In this scenario, a L1 contract would act like a proxy for the exit. Alice would specify that she is exiting 100 ETH_{L2} to the proxy contract address (instead of to her address) and set the proxy contract to forward it to her address (if/when it comes through after 7 days). Before the dispute window closes, she can sign a transaction instructing the proxy contract to forward the exit to Bob instead of to her (while giving Bob signing authority over it). In this way, the exit becomes tradeable. After 7 days, the current owner can ask the proxy to fetch the actual transfer from the bridge and forward it to them. If the exit fails, the bridge will refuse the exit.

Given this option, why modify the bridge/outbox of the rollup? This paper is not intended as a strong endorsement of either approach – the reader can decide between the two approaches. Our intention with this research is to discuss, design, implement, and measure the actual functionality of what is needed. This will be largely the same whether it is placed inside or outside the bridge/outbox. The main advantage of modifying the bridge/outbox is that is backward compatible with existing web3 bridge interfaces and with current user behaviour – if web3 interfaces or users do a slow withdraw, our solution can "bail them out" after the fact. Placing the functionality inside the bridge/outbox is more challenging in some regards (*e.g.,* existing code is complex to understand) but also easier in other regards (*e.g.,* our code has direct access to state variables). An outside contract might require minor changes to the bridge anyways, such as creating public interfaces to state variables or other data (*e.g.,* as one example, we later discuss how a prediction market must be able to query the outbox to know if an RBlock is pending, finalized, or failed, which is not a current feature). By contrast, the main advantage of an outside contract is modularity and reducing complexity (and thus risk) within the bridge.

3.3 Hedged Tradeable Exits

One remaining issue with tradeable exits is how specialized Bob is: he must have liquidity in ETH_{L1} (or worst, every token being withdrawn from L2), be online and active, know how to price derivatives, and be a L2 validator. While we can expect blockchain participants with each specialization, it is a lot to assume of a single entity. The goal of this subsection is to split Bob into two distinct participants: Carol and David. Our goal is to allow Carol who does not (or functionally cannot) know anything about L2's current state to safely accept a tradeable exit as if it were equivalent to finalized ETH_{L1} (or L1 tokens). Carol could be a L1 contract that accepts the withdrawn tokens for a service or enables exchange. In order to make Carol agnostic of L2, we need David to be aware of L2: David is a L2 validator who understands the risks of an RBlock failing and is willing to bet against it happening. Therefore David needs to also have some liquidity to bet with however it could be ETH_{L1} or a stablecoin, while Alice and Carol can interact with all sorts of tokens that David need not heard of or even ones David would not want to hold himself.

Recall that Alice wants ETH_{L1} quickly in order to do something on L1 with it; Carol can be that destination contract. The primary risk for Carol accepting ETH_{XX} as if it were ETH_{L1} is that the RBlock containing the ETH_{XX} withdrawal fails and the exit is worthless. If Alice can obtain insurance for the ETH_{XX} that can be verified via L1, then Carol's risk is hedged and she could accept ETH_{XX}. The insurance could take different forms but we propose using a prediction market.

Prediction markets

A decentralized prediction market is an autonomous (*e.g.,* vending machine-esque) third party contract. Since we are insuring L1 ETH_{XX}, we need to run the market on L1 (despite the fact that it would be cheaper and faster on L2). Consider a simple market structure based on [1]. A user can request that a new market is created for a given RBlock. The market checks the outbox for the RBlock and its current status (which must be pending). Once opened, any user can submit 1 ETH_{L1} (for example, the actual amount would be smaller but harder to read) and receive two "shares": one that is a bet that the RBlock will finalize, called $\text{FINAL}_{\mathsf{PM}}$, and one that is a bet that the RBlock will fail, called $\text{FAIL}_{\mathsf{PM}}$. These shares can be traded on any platform. At any time while the prediction market is open, any user can redeem 1 $\text{FINAL}_{\mathsf{PM}}$ and 1 $\text{FAIL}_{\mathsf{PM}}$ for 1 ETH_{L1}. Once the dispute period is over, any user can request that the market close. The market checks the rollup's outbox for the status of the RBlock– since both contacts are on L1, this can be done directly without oracles or governance. If the RBlock finalizes, it offers 1 ETH_{L1} for any 1 $\text{FINAL}_{\mathsf{PM}}$ (and conversely if it fails). The market always has enough ETH_{L1} to fully settle all outstanding shares.

It is argued in the prediction market literature [1] that (i) the price of one share matches the probability (according to the collective wisdom of the market) that its winning condition will occur, and (ii) the price of 1 $\text{FINAL}_{\mathsf{PM}}$ and 1 $\text{FAIL}_{\mathsf{PM}}$ will sum up to 1 ETH_{L1}. For example, if $\text{FAIL}_{\mathsf{PM}}$ trades for 0.001 ETH_{L1}, then (i) the market believes the RBlock will fail with probability of 0.1% and (ii) $\text{FINAL}_{\mathsf{PM}}$ will trade for 0.999 ETH_{L1}. These arguments do not assume market friction: if the gas cost for redeeming shares is D (for delivery cost), both share prices will incorporate D (see Section 5). Lastly, prediction markets are flexible and traders can enter and exit positions at any time – profiting when they correctly identify over- or under-valued forecasts. This is in contrast to an insurance-esque arrangement where the insurer is committed to hold their position until completion of the arrangement.

Hedging exits

Given a prediction market, Alice can hedge 100 ETH_{XX} by obtaining 100 $\text{FAIL}_{\mathsf{PM}}$ as insurance. Any autonomous L1 contract (Carol) should be willing to accept a portfolio of 100 ETH_{XX} and 100 $\text{FAIL}_{\mathsf{PM}}$ as a guaranteed delivery of 100 ETH_{L1} after the dispute period, even if Carol cannot validate the state of L2.

Perhaps surprisingly, this result collapses when withdrawing ETH_{L2}– consider Path 1 through the protocol. Alice withdraws 100 ETH_{L2} from L2 and obtains 100 ETH_{XX}. Bob creates 100 $\text{FAIL}_{\mathsf{PM}}$ and 100 $\text{FINAL}_{\mathsf{PM}}$ for a cost of 100 ETH_{L1}. Alice buys 100 $\text{FAIL}_{\mathsf{PM}}$ from Bob for a small fee. Alice gives Carol 100 ETH_{XX} and 100 $\text{FAIL}_{\mathsf{PM}}$ and is credited as if she deposited 100 ETH_{L1}. In seven days, Bob gets 100 ETH_{L1} for his 100 $\text{FINAL}_{\mathsf{PM}}$ and Carol gets 100 ETH_{L1} for her 100 ETH_{XX}. If the RBlock fails, Bob has 0 ETH_{L1} and Carol has 100 ETH_{L1} from the 100 $\text{FAIL}_{\mathsf{PM}}$. In both cases, Alice has a balance of 100 ETH_{L1} with Carol.

In path 2, Alice withdraws 100 ETH_{L2} from L2 and obtains 100 ETH_{XX}. Alice sells 100 ETH_{XX} to Bob for 100 ETH_{L1}. Alice gives Carol 100 ETH_{L1} and is credited with a balance of 100 ETH_{L1}. In 7 days, Bob gets 100 ETH_{L1} for his 100 ETH_{XX} and Carol has 100 ETH_{L1}. If the RBlock fails, Bob has 0 ETH_{L1}, Carol has 100 ETH_{L1}, and Alice has a balance of 100 ETH_{L1} with Carol.

Modulo differing gas costs and market transaction fees, paths 1 and 2 are equivalent. Path 2 does not use a prediction market at all, it only uses basic tradeable exits. Given this, do prediction markets add nothing to tradeable exits? We argue prediction markets still have value for a few reasons. (1) Speculators will also participate in the prediction market which gives Alice a chance for a fast exit even without Bob (an L2 validator). (2) If Alice withdraws a token other than ETH, the prediction market should still be set up to payout in ETH (otherwise you end up with 50 separate prediction markets for the 50 different kinds of tokens in any given RBlock). In this case, Alice can obtain FAIL_{PM} when Bob has no liquidity or interest in the token she is withdrawing (however Carol needs to incorporate an exchange rate risk when accepting an exit in one token and the insurance in ETH). (3) The PM can also help with NFTs and other non-liquid tokens (see Section 6.4).

Three of the most common types of traders are utility traders, speculators, and dealers [6]. With a prediction market, Alice is a utility trader and Bob is a dealer. However, there might exist speculators who want to participate in the market because they have forecasts about rollup technology, a given RBlock, the potential for software errors in the rollup or in the validator software, *etc.* Executives of rollup companies could receive bonuses in FINAL_{PM}. Quick validators might profit from noticing an invalid RBlock with FAIL_{PM} or they might be betting on an implementation bug or weeklong censorship of the network. Speculators add liquidity to the prediction market which reduces transactional fees for Alice. However, speculation also brings externalities to the rollup system where the side-bets on an RBlock could exceed the staking requirements for posting an RBlock, breaking the crypo-economic arguments for the rollup. In reality, these externalities can never be prevented in any decentralized incentive-based system [3].

4 Implementation and Performance Measurements

We run *Arbitrum Nitro* test-net locally and use Hardhat [4] for our experiments. We obtain our performance metrics using TypeScripts scripts.

4.1 Tradeable Exits

Trading the exit directly through the bridge/outbox

We fork the *Arbitrum Nitro* outbox to add native support for tradeable exits. The modified outbox is open source, written in 294 lines (SLOC) of Solidity, and a bytecode of 6,212 bytes (increased by 1,197 bytes). The solidity code and Hardhat scripts are available in a GitHub repository.[4] Our modifications include:

- Adding the `transferSpender()` function which allows the exit owner to transfer the exit to any L1 address even though the dispute period is not passed.
- Adding the `isTransferred()` mapping which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is a boolean.

[4] GitHub:Nitro, Fast-Withdrawals: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals`

- Adding the `transferredToAddress` mapping which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is the current owner of the exit.
- Modifying the `executeTransactionImpl()` function. Once the dispute period is passed and the withdrawal transaction is confirmed, anyone can call the `executeTransaction()` function from the outbox (which internally calls the `executeTransactionImpl()`) and release the funds to the account that was specified by the user 7 days earlier in the L2 withdrawal request. With our modifications, this function is now enabled to release the requested funds to the current owner of the exit.

To execute the `transferSpender()` function; Alice (who has initiated a withdrawal for 100 ETH_{L2}) has to provide variables related to her exit (*e.g.,* exit number), which she can query using the Arbitrum SDK[5], as well as the L1 address she wants to transfer her exit to. The `transferSpender()` function then checks (1) if the exit is already spent, (2) it is already transferred, and (3) the exit is actually a leaf in any unconfirmed RBlock. If the exit has been transferred, the `msg.sender` is cross-checked against the current owner of the exit (recall exit owners are tracked in the `transferredToAddress` mapping added to the outbox). Once these tests are successfully passed, the `transferSpender()` function updates the exit owner by changing the address in the `transferredToAddress` mapping. This costs 85,945 units of L1 gas. Note that the first transfer always costs more as the user has to pay for initializing the `transferredToAddress` mapping. `transferSpender()` costs 48,810 and 48,798 units of L1 gas for the second and third transfer respectively. The `gasUed` for executing the new `executeTransactionImpl()` function is 91,418 units of L1 gas.

Trading the exit through an L1 market

We also implement and deploy an L1 market that allows users to trade their exits on L1 even though the dispute window is not passed (see Section 6.3 for why *Uniswap* is not appropriate). In addition, we add a new function to the *Arbitrum Nitro* outbox, the `checkExitOwner()`, which returns the current owner of the exit. Figure 1 illustrates an overview of participant interactions and related gas costs. To start trading, Alice needs to lock her exit up in the market by calling the `transferSpender()` function from the outbox. Next, she can open a market on this exit by calling the `openMarket()` from the market contract and providing the ask price. The market checks if Alice has locked her exit (by calling the `checkExitOwner()` from the outbox) and only in that case a listing is created on this exit. The market would be open until a trade occurs or Alice calls the `closeMarket()` on her exit. Bob, who is willing to buy Alice's exit, calls the payable `submitBid()` function from the market contract. If the `msg.value` is equal or greater than Alice's ask price, the trade occurs; (1) the market calls the `transferSpender()` from the outbox providing Bob's address. Note that market can only do that since it is the current owner of the exit being traded, and (2) the `msg.value` is transferred to Alice.

The market and modified outbox are open source and written in 125 and 294 lines (SLOC) of Solidity respectively. The solidity code for these contracts in addition to the Hardhat scripts are available in a GitHub repository.[6] Once deployed, the bytecode of the market and outbox is 5,772 and 6,264 bytes respectively.

[5] A typescript library for client-side interactions with Arbitrum.
[6] GitHub:Nitro, Fast-Withdrawals: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals`

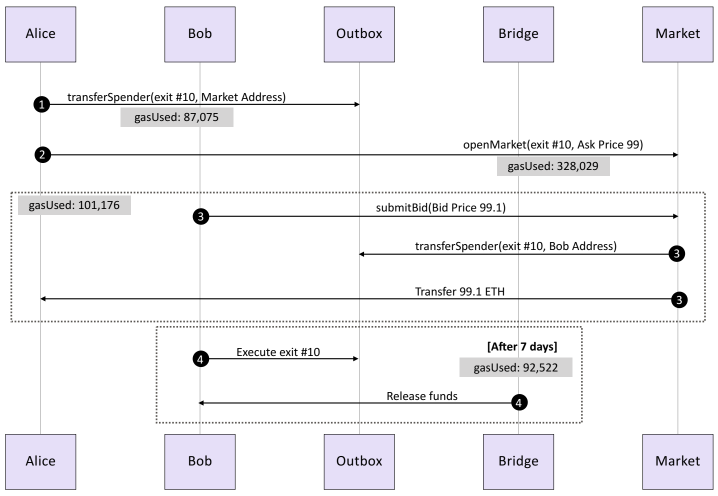


Figure 1 Overview of trading the exit through an L1 market.

4.2 Prediction Market

As described in Section 3.3, a prediction market can be used to hedge the exit. We do not implement this as one can use an existing decentralized prediction market (*e.g., Augur* or *Gnosis*). However, we further modify *Arbitrum Nitro* to make it friendly to a prediction market that wants to learn the status of an RBlock (pending, confirmed). More specifically, we modify the *Arbitrum Nitro* outbox and RollupCore smart contracts, modifications include:

- Adding the `assertionAtState` mapping to the outbox which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is the user-defined data type `state` that restricts the variable to have only one of the `pending and confirmed` predefined values.
- Adding the `markAsPending` function to the outbox which accepts an RBlock and marks it as pending in the `assertionAtState` mapping.
- Adding the `markAsConfirmed` function to the outbox which accepts an RBlock and marks it as confirmed in the `assertionAtState` mapping.
- Modifying the `createNewNode()` function in the RollupCore contract. To propose an RBlock, the validator acts through the RollupCore contract by calling a `createNewNode()` function. We modify this function to call the `markAsPending()` from the outbox which marks the RBlock as pending.
- Modifying the `confirmNode()` function in the RollupCore contract. Once an RBlock is confirmed, the validator acts through the RollupCore contract via `confirmNode` to move the now confirmed RBlock to the outbox. We modify this function to call the `markAsConfirmed()` from the outbox which marks the RBlock as confirmed.

The modified outbox and RollupCore are open source and written in 297 and 560 lines (SLOC) of Solidity respectively. The solidity code for these contracts in addition to the Hardhat scripts are available in a GitHub repository.[7] Once deployed, the bytecode of the outbox and RollupCore is 6,434 and 3,099 bytes respectively.

[7] GitHub:Nitro, Fast-Withdrawals: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals`

5 Pricing

Pricing ETH$_{\mathsf{XX}}$

Consider how much you would pay for 100 ETH$_{\mathsf{XX}}$ (finalized in 7 days = 168 hours) in ETH$_{\mathsf{L1}}$ today. Since ETH$_{\mathsf{XX}}$ is less flexible than ETH$_{\mathsf{L1}}$, it is likely that you do not prefer it to ETH$_{\mathsf{L1}}$, so our intuition is that it should be priced less (*e.g.,* 100 ETH$_{\mathsf{XX}}$ = 99 ETH$_{\mathsf{L1}}$). However, our solution works for any pricing and we can even contrive corner cases where ETH$_{\mathsf{XX}}$ might be worth more than ETH$_{\mathsf{L1}}$ by understanding the factors underlying the price.

In traditional finance [7], forward contracts (and futures, which are standardized, exchange traded forwards) are very similar to ETH$_{\mathsf{XX}}$ in that they price today the delivery of an asset or commodity at some future date. One key difference is that with a forward contract, the price is decided today but the actual money is exchanged for the asset at delivery time. When ETH$_{\mathsf{XX}}$ is sold for ETH$_{\mathsf{L1}}$, both price determination and the exchange happen today, while the delivery of ETH$_{\mathsf{L1}}$ for ETH$_{\mathsf{XX}}$ happens in the future. The consequence is that we can adapt pricing equations for forwards/futures, however, the signs (positive/negative) of certain terms need to be inverted.

We review the factors [7] that determine the price of a forward contract (F_0) and translate what they mean for ETH$_{\mathsf{XX}}$:

- *Spot price of ETH$_{\mathsf{L1}}$ (S_0):* the price today of what will be delivered in the future. ETH$_{\mathsf{XX}}$ is the future delivery of ETH$_{\mathsf{L1}}$, which is by definition worth 100 ETH$_{\mathsf{L1}}$ today.
- *Settlement time (Δt):* the time until the exit can be traded for ETH$_{\mathsf{L1}}$. In *Arbitrum*, the time depends on whether disputes happen. We simplify by assuming Δt is always 7 days (168 hours) from the assertion time. A known fact about forwards is that F_0 and S_0 converge as Δt approaches 0.
- *Storage cost (U):* most relevant for commodities, receiving delivery of a commodity at a future date relieves the buyer of paying to store it in the short-term. Securing ETH$_{\mathsf{XX}}$ and securing ETH$_{\mathsf{L1}}$ is identical in normal circumstances, so not having to take possession of ETH$_{\mathsf{L1}}$ for Δt time does not reduce costs for a ETH$_{\mathsf{XX}}$ holder.
- *Delivery cost (D):* the cost of delivery of the asset, which in our case will encompass gas costs. Exchanging ETH$_{\mathsf{L1}}$ for ETH$_{\mathsf{XX}}$ requires a transaction fee and also creates a future transaction fee to process the exit (comparable in cost to purchasing a token from an automated market maker). An ETH$_{\mathsf{L1}}$ seller should be compensated for these costs in the price of ETH$_{\mathsf{XX}}$.
- *Exchange rate risk:* a relevant factor when the asset being delivered is different than the asset paying for the forward. In our case, we are determining the price in ETH$_{\mathsf{L1}}$ for future delivery of ETH$_{\mathsf{L1}}$, thus, there is no exchange risk at this level of the transaction. However, the price of gas (in the term D) is subject to ETH/gas exchange rates. For simplicity, we assume this is built into D.
- *Interest / Yield ($-r + y$):* both ETH$_{\mathsf{L1}}$ and ETH$_{\mathsf{XX}}$ have the potential to earn interest or yield (compounding over Δt), while for other tokens, there might be an opportunity to earn new tokens simply by holding the token. Let r be the (risk-free) interest (yield) rate for ETH$_{\mathsf{L1}}$ that cannot be earned by ETH$_{\mathsf{XX}}$, while y is the opposite: yield earned from ETH$_{\mathsf{XX}}$ and not ETH$_{\mathsf{L1}}$. Initially $y > 1$ and $r = 0$, however, with ETH$_{\mathsf{XX}}$ becoming mainstream, it is possible $r = y$ (especially hedged ETH$_{\mathsf{XX}}$).
- *Settlement risk (R):* the probability that ETH$_{\mathsf{L1}}$ will fail to be delivered for ETH$_{\mathsf{XX}}$ discounts the price of ETH$_{\mathsf{XX}}$. We will deal with this separately.

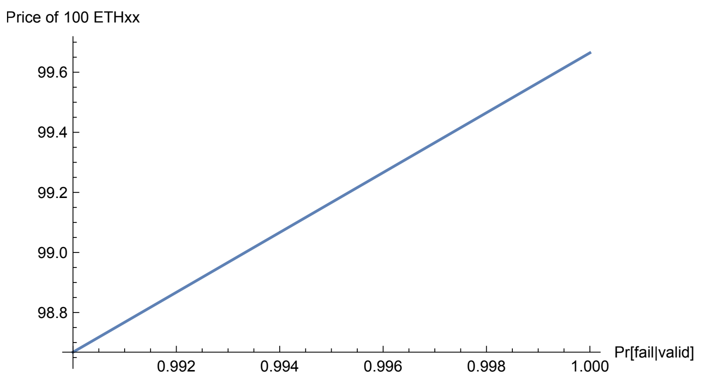


Figure 2 Price of 100 ETH_{XX} (in ETH) as the probability an RBlock actually finalizes (given the validator checks it with software validation) varies from 99% to 100%, which is denoted by R. Note that 99% is an extraordinarily low probability for this event (considering an RBlock has never failed at the time of writing). The take-away is that the price is not very sensitive to how precisely we estimate R.

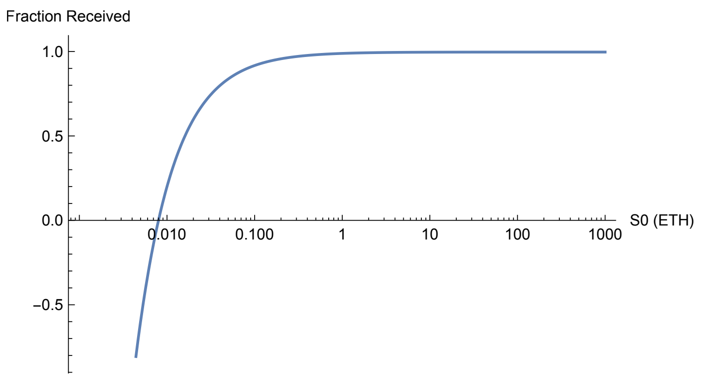


Figure 3 This chart shows the percentage of ETH recovered (F_0/S_0) as the amount withdrawn (S_0) increases (log scale), demonstrating it is only economical for withdrawing larger amounts of ETH_{L2}. At low values, the gas costs of a withdrawal dominate. At very low values, the gas costs exceed the price of ETH_{XX} causing the curve to go negative.

Put together, the price of ETH_{XX} (F_0) is:

$$F_0 = (S_0 + U - D) \cdot e^{(-r+y)\cdot \Delta t} \cdot R$$

This value, F_0, is an expected value – the product of the value and the probability that the RBlock fails to finalize. However, the trader is informed because they have run verification software and checked that the RBlock validates.

$$R = (1 - \mathbf{Pr}[\text{rblock fails to finalize}|\text{rblock passes software verification}])$$

Working Example

We start with R. For an RBlock to be up for consideration, it must be submitted to the outbox as a potential solution and for it to fail, a dispute must be filed with an alternative RBlock that the L1 outbox deems to be correct. In our case, the buyer of ETH_{XX} actually runs a L2 validator and thus performs software validation on the RBlock, and will not accept it if the software does not validate it. For an RBlock to fail given the software validation, it software must have an error that causes a discrepancy between it and the L1 outbox. Furthermore, at least one other validator would need to have different, correct software, and this validator would need to be paying attention to this specific RBlock and independently check it. This should be a rare event and assume $R = (1 - 10^{-15})$ for this example. Figure 2 shows a range of R values.

Next, consider the resulting price of F_0. Alice starts with 100 ETH_{XX} and Bob purchases it from her. Bob can hold ETH_{XX} with no cost ($U = 0$). Alice pays the transaction fee for the deposit, however the cost for the contract for exiting ETH_{XX} into ETH_{L1} after the dispute period is expected to be $D = 0.008$ ETH (D). Assume a safe-ish annual percent yield (APY) on ETH deposits is 0.2%. Assume ETH_{XX} expires in 6 days (0.0164 years). ETH_{XX} earns no yield ($y = 0$). Plugging this into the equation, $F_0 = 99.665$ ETH.

As a second example, consider a smaller amount like 0.05 ETH_{XX} (less than \$100 USD at time of writing). Now the gas costs are more dominating. $F_0 = 0.04186$ ETH_{L1} which is only 83.7%. This demonstrates that fast exits are expensive for withdrawals of amounts in the hundreds of dollars. Figure 3 shows a range of withdraw amounts.

Lastly, could ETH_{XX} ever be worth more than ETH_{L1}? The equation says yes: with a sufficiently high U or y. A contrived example would be some time-deferral reason (*e.g.,* tax avoidance) to prefer receiving ETH_{L1} in 7 days instead of today. However, in order to purchase ETH_{XX} at a premium to ETH_{L1}, it would have to be cheaper to trade for it than to simply manufacture it. Someone holding ETH_{L1} and wanting ETH_{XX} could simply move it to L2 and then immediately withdraw it to create ETH_{XX}. The gas cost of this path will be one upper bound on how much ETH_{XX} could exceed ETH_{L1} in value.

Pricing FINAL_{PM} and FAIL_{PM}

It might appear surprising at first, but one of the main results of this paper is that the price of 100 ETH_{XX} and the of price 100 FINAL_{PM} are essentially the same. Both are instruments that are redeemable at the same future time for the same amount of ETH_{L1} (either 100 if the RBlock finalizes and 0 if the RBlock fails) with the same probability of failure (that the RBlock fails). The carrying costs of both are identical. There may be slight differences in the gas costs of redeeming ETH_{L1} once the dispute period is over. However, the operation (at a computational level) is largely the same process. This is actually a natural result: if 100 FAIL_{PM} perfectly hedges (reduces the risk to zero) the failure of 100 ETH_{XX} to finalize, then the compliment to FAIL_{PM}, FINAL_{PM}, should be priced the same as ETH_{XX}.

6 Discussion

6.1 Prediction Market Fidelity

A prediction market that covers a larger event should attract more interest and liquidity. For example, betting on an entire RBlock will have more market interest than betting on Alice's specific exit. On the other hand, if markets are exit-specific, the market can be established immediately after Alice's withdrawal hits the inbox instead of waiting for an RBlock (hence $\sim$ in Table 1 to indicate it could be done within one L1 transaction). Another consideration arrises when tokens other than ETH are being withdrawn – if the payout of the market matches the withdrawn token, FAIL_{PM} will perfectly hedge the exit. Otherwise the hedge is in the equivalent amount of ETH which could change over 7 days. Our suggestion is to promote the most traders in a single market and avoid fragmentation – so we suggest one market in one payout currency (ETH) for one entire RBlock.

6.2 Withdrawal Format

As implemented, transferable exits can only be transferred in their entirety. If Alice wants to withdraw 100 ETH_{L2} and give 50 ETH_{XX} to one person and 50 ETH_{XX} to another, she cannot change this once she has initiated the withdraw (if she anticipates it, she can request two separate withdrawals for the smaller amounts). We could implement divisible exits and for ETH; there are no foreseen challenges since the semantics of ETH_{L1} are specified at the protocol-level of Ethereum. However for custom tokens, the bridge would need to know how divisible (if at all) a token is. In fact, a bridge should ensure that the L2 behavior of the tokens is the same as L1 (or that any inconsistencies are not meaningful). Even if a token implementation is standard, such as ERC20, this only ensures it realizes a certain interface (function names and parameters) and does not mean the functions themselves are implemented as expected (parasitic ERC20 contracts are sometimes used to trick automated trading bots.[8] The end result is that bridges today do not allow arbitrary tokens; they are built with allowlists of tokens that are human-reviewed and added by an authorized developer. In this case, ensuring divisible exits are not more divisible than the underlying token should be feasible, but we have not implemented it.

6.3 Markets

At the time of writing, the most common way of exchanging tokens on-chain is with an automated market maker (AMM) (*e.g., Uniswap*). If Alice withdraws ETH_{XX} and Bob is a willing buyer with ETH_{L1}, an AMM is not the best market type for them to arrange a trade. AMMs use liquidity providers (LPs) who provide both token types: Alice has ETH_{XX} but no ETH_{L1} that she is willing to lock up (hence why she is trying to fast exit). Bob has ETH_{L1} but to be an LP, he would also need to have ETH_{XX} from another user. However, this only pushes the problem to how Bob got ETH_{XX} from that user. The first user to sell ETH_{XX} cannot use an AMM without locking up ETH_{L1}, which is equivalent to selling ETH_{XX} to herself for ETH_{L1}. The second challenge of an AMM is the unlikely case that an RBlock fails and ETH_{XX} is worthless – then the LPs have to race to withdraw their collateral before other users extract it with worthless ETH_{XX}. It is better to use a traditional order-based market; however, these are expensive to run on L1 [11]. One could do the matchmaking on L2 and then have the buyer and seller execute on L1, but this reintroduces the griefing attacks we have tried to avoid. For now, we implement a very simple one-sided market where Alice can deposit her ETH_{XX} and an offer price, and Bob can later execute the trade against. If Alice is unsure how to price ETH_{XX}, an auction mechanism could be used instead.

6.4 Low Liquidity or Non-Fungible Tokens

For tokens that have low liquidity on L1, or in the extreme case, are unique (*e.g.,* an NFT), fast exits do not seem feasible. All the fast exit methods we examined do not actually withdraw the original tokens faster; they substitute a functionally equivalent token that is already on L1. However, we can still help out with low-liquidity withdrawals. We should consider *why* the user wants a fast exit. If it is to sell the token, they can sell the exit instead of the token to any buyer that is L2-aware and willing to wait 7 days to take actual possession. To sell to an L2-agnostic buyer, the seller can insure the exit with enough $\text{FAIL}_{\mathsf{PM}}$ to cover the purchase price. In this case, the buyer does not get the NFT if the RBlock fails but they get their money back.

[8] "Bad Sandwich: DeFi Trader 'Poisons' Front-Running Miners for $250K Profit." *Coindesk*, Mar 2021.

7 Concluding Remarks

This paper addresses a common "pain point" for users of L2 optimistic rollups on Ethereum. The 7-day dispute period prevents users from withdrawing ETH, tokens, and data quickly. Tradeable exits provide users with flexibility after they request a withdrawal. If they decide 7 days is too long, they can seek to trade their exit for ETH_{L1} or they can ask a contract to accept their ETH_{XX} by bundling it with insurance against the failure of the RBlock– this way the contract does not have to be L2-aware. While some users might still prefer the features of other withdrawal methods (centralized exchanges or solution like *Hop*), it is useful to make the native rollup functionality as flexible as possible, especially for users who do not realize that a withdrawal induces a 7-day waiting period until it is too late.

References

1 Jeremy Clark, Joseph Bonneau, Edward W Felten, Joshua A Kroll, Andrew Miller, and Arvind Narayanan. On decentralizing prediction markets and order books. In *Workshop on the Economics of Information Security (WEIS)*, volume 188, 2014.

2 Didem Demirag and Jeremy Clark. Absentia: Secure multiparty computation on ethereum. In *Workshop on Trusted Smart Contracts (WTSC)*, pages 381–396. Springer, 2021.

3 Bryan Ford and Rainer Böhme. Rationality is self-defeating in permissionless systems. Technical Report cs.CR 1910.08820, arXiv, 2019.

4 Nomic Foundation. Hardhat. `https://hardhat.org`, October 2022. (Accessed on 10/18/2022).

5 Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *Financial Cryptography*, 2020. `doi:10.1007/978-3-030-51280-4_12`.

6 Larry Harris. *Trading and exchanges: market microstructure for practitioners*. Oxford, 2003.

7 John Hull, Sirimon Treepongkaruna, David Colwell, Richard Heaney, and David Pitt. *Fundamentals of futures and options markets*. Pearson Higher Education AU, 2013.

8 Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*, pages 1353–1370, 2018.

9 Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. Sok: Validating bridges as a scaling solution for blockchains. Technical report, Cryptology ePrint Archive, 2021.

10 Sarah Meiklejohn. An evolution of models for zero-knowledge proofs. In *EUROCRYPT (invited talk)*, 2021.

11 Mahsa Moosavi and Jeremy Clark. Lissy: Experimenting with on-chain order books. In *Workshop on Trusted Smart Contracts (WTSC)*, 2021.

12 Paul Sztorc. Truthcoin. Technical report, Online, 2015.

13 Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022.

14 Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. Sok: Communication across distributed ledgers. In *Financial Cryptography*, 2021.

Buying Time: Latency Racing vs. Bidding for Transaction Ordering

Akaki Mamageishvili
Offchain Labs, Zürich, Switzerland

Mahimna Kelkar[1]
Cornell University, New York, NY, USA

Jan Christoph Schlegel
City, University of London, UK

Edward W. Felten
Offchain Labs, Washington, D.C., USA

Abstract

We design TimeBoost: a practical transaction ordering policy for rollup sequencers that takes into account both transaction timestamps and bids; it works by creating a score from timestamps and bids, and orders transactions based on this score.

TimeBoost is transaction-data-independent (i.e., can work with encrypted transactions) and supports low transaction finalization times similar to a first-come first-serve (FCFS or pure-latency) ordering policy. At the same time, it avoids the inefficient latency competition created by an FCFS policy. It further satisfies useful economic properties of first-price auctions that come with a pure-bidding policy. We show through rigorous economic analyses how TimeBoost allows players to compete on arbitrage opportunities in a way that results in better guarantees compared to both pure-latency and pure-bidding approaches.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Transaction ordering, First-come-first-serve, First-price auctions

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.23

Related Version *Full Version*: `https://arxiv.org/abs/2306.02179`

Acknowledgements We are grateful to Lee Bousfield, Chris Buckland, Potuz Heluani, Raul Jordan, Mallesh Pai, Ron Siegel, Terence Tsao as well as participants at the Swiss National Bank Technology and Finance Seminar for interesting discussions and valuable feedback.

1 Introduction

Transaction ordering is critically important for financial systems – the order in which user transactions are executed can directly impact the profits made by users. This motivates the study of designing transaction ordering policies with useful properties.

In this work, we focus on ordering policies for *centralized sequencers* – meaning that a single *sequencer* receives transactions from users and publishes an ordered sequence to be used for execution. A transaction ordering policy here specifies how the resulting output sequence depends on the contents and arrival times of transactions at the sequencer. Our work provides rigorous economic analyses to justify the utility of our proposed policy.

[1] This work was completed in the author's role at Offchain Labs.

Why consider a centralized sequencer? In addition to the centralized sequencer setting being a potentially simpler model to study as a first step, there are two other main reasons why we choose to do so in this work:

1. *Existing use-cases are already centralized.* Decentralized blockchains such as Ethereum are still *ephemerally centralized* with respect to ordering – for a given block, similar to a centralized sequencer, only a single miner/validator is in complete control of the inclusion and ordering of transactions within the block. Similarly, current layer-2 "rollup" protocols (such as Arbitrum and Optimism) also employ a centralized sequencer to order transactions in a batch posted to the underlying Ethereum base-chain.
2. *Ordering policies are mostly orthogonal to the problem of sequencer decentralization.* While decentralizing the sequencer is an important active research direction, we note that a suitable transaction ordering policy can be chosen orthogonally to the method of sequencer decentralization. In particular, the decentralized protocol can first be used to agree on single *pre-ordering* or *scoring* of transactions, following which a specific ordering policy can be applied. In other words, the output of the decentralized protocol can be thought of simulating the input of a virtual centralized sequencer on which the ordering policy gets applied.
   An example of this is seen in the recent line of works on fair-ordering [3,8,9,11,20] – they can be thought of as a decentralized implementation of a first-come-first-serve ordering policy which combines local transaction orderings from many nodes.

Furthermore, while current centralized sequencer implementations are semi-trusted in that they receive transactions in plaintext and are expected not to deviate from the specified ordering policy or insert transactions of their own, we note that transaction data can be hidden from the sequencer by using threshold decryption by a committee (i.e., the sequencer only sees encrypted transactions and orders them, only after which a committee decrypts the plaintext) or trusted hardware (such as Intel SGX). Through these techniques, the adversarial behavior of the sequencer can be substantially restricted.

The study of ordering policies is important even when the sequencer is trusted (or is suitably constrained as mentioned above) due to the presence of other profit-seeking entities in the system. For instance, after the sequencer publishes state after execution of previous transaction(s), arbitrage opportunities can be created; players in the system will compete with each other to take advantage of these opportunities. Similar situations can also arise due to state updates from external systems.

1.1 Existing Ordering Policies

Ordering policies used on blockchains today fall roughly into three categories described below.

First-come first-serve (FCFS). One natural ordering policy is the first-come, first-serve (FCFS) rule. Here, transactions are sequenced in the same order that they were received from users. There are several advantages to FCFS: to begin, it is simple to implement and seems intuitively fair – after all, it is a commonly used policy even for real-world interactions. FCFS also minimizes transaction latency: transactions can be continuously sequenced as they arrive, and do not need to conform to the discrete granularity of blocks. The sequencer in the layer-2 rollup Arbitrum employs an FCFS policy.

One major disadvantage of FCFS however, is that creates *latency competition* in the sense that entities are incentivized to position themselves as close to the sequencer as possible in order to be the first to react to any new market information. This is a well known and studied problem within traditional financial systems. Indeed, high frequency trading (HFT) firms invest millions of dollars into low-latency infrastructure that can operate sub-microsecond or even finer scales; their trading accounts for roughly half of all trading volume [13]. This inclination to latency investment is highly inefficient since the investment happens externally to the system (as opposed to bidding; see below) and therefore cannot be used beneficially within the system. Recent works [1, 17] have also shown the potential for similar strategic manipulation within a pure FCFS protocol in the decentralized setting.

One crucial point to emphasize here is that this latency competition in FCFS *does not disappear even if transaction data is hidden* (e.g., transactions are encrypted). This is because any state changes (from the sequencer or even from external systems) can trigger a profit opportunity wherein it is beneficial to have the quickest access to the sequencer. As a specific example, an update on the trading price of a token can create an arbitrage opportunity whose profit will go only to the player who can submit its transaction to the sequencer first[2]. This kind of latency-based arbitrage has already been seen in Arbitrum, which implements a centralized FCFS sequencer.

Per-block transaction bidding. A second natural policy is to group transactions into blocks, then order transactions within a block based on their *bid*. Specifically, each transaction is submitted along with a fee or bid; the sequencer now collects all transactions submitted within some time interval and sequences them by the descending order of their bids. This essentially simulates a first-price all-pay auction [10] (i.e., players bid independently; the highest bid wins but all players need to pay their bid amount) to take advantage of a particular arbitrage opportunity. Since players submit their bids independently, the bidding policy can work as expected even when transactions are encrypted (since state or market updates create arbitrage opportunities).

One advantage of a bidding policy (compared to FCFS) is that the payment is internal to the system and therefore can be utilized within it to e.g., subsidize protocol operation costs.

When the block-time is large (e.g., $12s$ as in Ethereum), it is expected that for almost all arbitrage opportunities, all interested players can post their bid within the time interval in an attempt to take advantage of the opportunity. However, when the block-time is small (this is typically the case in layer-2 protocols to increase scalability), perhaps surprisingly, having a connection with lower latency can provide a substantial advantage. This is because when the market update happens close to end of the block time, only players with a faster connection will be able get their transaction included in the block; consequently, they may be able to take advantage of the arbitrage opportunity with a smaller (or even a zero) bid.

Looking ahead, our TimeBoost policy (which combines both arrival times and bidding) will enable arbitrageurs to prefer bidding even when block times are small, thereby allowing the protocol to capture this value rather than it being lost to external latency infrastructure.

Block or MEV auctions. A third widely-used policy auctions off the complete rights to choose and order transactions within a block. Here, the sequencer does not order transactions itself but rather accepts block proposals from external players (often called block *builders*) and chooses the proposal from the builder who pays the most. These auctions initially arose from the realization that significant profit (often referred to as maximal (previously miner) extractable value or MEV [2, 6]) can be extracted by manipulating the ordering of user

[2] Another approach if the sequencer broadcasts state information in a random order to clients is to create many dummy client copies, thereby increasing the chances that some copy gets the feed faster.

transactions. In the past two years, through companies such as Flashbots and Bloxroute, an MEV marketplace has been created on Ethereum *outside of the protocol* to connect block proposers (entities in charge of proposing or sequencing a block) to block builders (players who find MEV opportunities and order user transactions to take advantage of them) – the result has been the extraction of hundreds of millions of dollars in profit from user transactions [15, 18].

While some MEV (such as arbitrage, which provides incentives for price discovery) is benign and can be done without the knowledge of user transactions, other forms of MEV extraction crucially rely on the transaction data. Recent works [18, 19] have shown such MEV to be significantly detrimental to users. The emergence of such MEV extraction has largely been attributed to the rationality of block proposers as well as the lack of regulation. For example, in traditional financial systems, it is often illegal or at the very least heavily constrained to profit from the knowledge of user transactions (for instance, payment-for-order-flow (PFOF): the selling of user transaction data is illegal in the UK, and, while legal in the US, still requires users to be provided with guarantees of "best execution").

A design goal for our work is therefore to design ordering policies that are data-independent, i.e., they do not use transaction data for ordering. This will allow them to be used even when transactions are encrypted at the time of sequencing.

1.2 Our contributions

TimeBoost: An ordering policy that combines FCFS and bidding. We propose TimeBoost, an ordering policy that combines both FCFS-style timestamps and first-price auction style bids. Below, we describe several natural goals that went into our design.

1. **Data independence.** The policy should not utilize the transaction data for ordering. This is a natural goal in order to support encrypted transactions and prevent data-dependent MEV attacks on transaction ordering.
2. **Low finalization time.** The policy should be able to sequence transactions within a short time g (the specific parameter can be set according to the application). This is important to improve the user experience with the system since transactions will be sequenced within time g after they are received.
3. **Independence of irrelevant transactions.** The ordering between two given transactions should not depend on the presence of other transactions. This is useful to prevent an adversary from inserting irrelevant transactions that results in flipping the ordering between two target transactions. Importantly, this property also ensures that a transaction submitter's strategy need only consider transactions that are relevant to the party's goals – for example if Alice is trying to capture a particular arbitrage opportunity, she need only worry about other transactions affecting that opportunity.
4. **Inclination to spending via bids instead of latency infrastructure.** As mentioned before, investments into latency infrastructure are highly inefficient from the system standpoint since the value spent cannot be utilized effectively by the system. Therefore, a natural goal is to disincentivize latency investment and instead incentivize players to bid for their transactions. Looking ahead, perhaps surprisingly, we find that the pure bidding policy results in a larger latency competition than our TimeBoost policy which combines bidding with FCFS style timestamps.

TimeBoost details. Intuitively, TimeBoost works by assigning *scores* to transactions based on both their arrival times and their bid. The final ordering is taken to be descending in the transaction scores. More specifically, for a transaction with arrival time t and bid b, TimeBoost assigns it the score $S(\mathsf{tx}) = \pi(b) - t$ where intuitively π represents a function for "buying time" – by increasing the transaction bid, users can reduce their effective timestamp (or equivalently, increase their score). Section 3 describes how to choose the function π.

Importantly, there is a limit to how much time can be "bought" through the bid – in particular, no transaction can outbid a transaction received some g time earlier. Such a property is required to ensure the quick finalization of user transactions. At the same time, transactions received less than g time before can always be outbid; this means that arbitrageurs always have g time to compete for any arbitrage opportunity as opposed to a pure bidding policy and will therefore prefer bidding over latency infrastructure investments.

We also show that TimeBoost satisfies all the useful economic properties of first-price all-pay auctions. Further, we show that players spend exactly the same amount in total with TimeBoost, as they would spend if only latency investment was allowed, except that most of the investment is done through bidding and therefore can be captured within the protocol for e.g., lowering user fees or for protocol development.

2 Ordering Policies

2.1 Preliminaries

A transaction tx that arrives at the sequencer can be characterized by a tuple (data, t, b) where data represents the transaction data, t denotes the arrival time, and b denotes the transaction bid (note that when transactions are of different sizes, b can be instead be considered to be a bid per unit size). Let $\mathcal{T}$ denote the set of all possible transactions; in principle this can be infinite or even uncountable (e.g., if arrival times are in $\mathbb{R}^+$) and our results do hold for these cases. For practical use-cases, typically, arrival times can be assumed to be in $\mathbb{Q}^+$ and bids can be assumed to be in $\mathbb{N}^{\geq 0}$.

An ordering policy now defines how a sequencer orders a finite set $\mathcal{T}'$ of transactions that it has received. A formal definition is given below:

▶ **Definition 1** ((Data-Independent) Ordering Policy)**.** *An* ordering policy *(or algorithm)* $\mathbb{P}$ *takes as input a finite subset* $\mathcal{T}' \subseteq \mathcal{T}$ *of transactions and outputs a linear ordering* $\mathbb{P}(\mathcal{T}')$*. For* $\mathsf{tx} \in \mathcal{T}'$*, let* $\mathbb{P}(\mathcal{T}', \mathsf{tx})$ *denote the position of transaction* tx *in the ordering* $\mathbb{P}(\mathcal{T}')$*. In other words, given* $\mathcal{T}'$ *and* $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}'$*,* $\mathbb{P}$ *outputs* tx_a *before* tx_b *if* $\mathbb{P}(\mathcal{T}', \mathsf{tx}_a) < \mathbb{P}(\mathcal{T}', \mathsf{tx}_b)$*.*

A policy is further called data-independent if it does not make use of the transaction data (i.e., it only uses the arrival time and the bid).

Since we want our ordering policies to not be based on the transaction content, we only consider data-independent policies for the rest of the paper. For simplicity, we can therefore represent a transaction tx simply by the tuple $(\mathsf{tx}.t, \mathsf{tx}.b)$. Furthermore, since ties can be broken by some chosen technique, without loss of generality, we can also assume $(\mathsf{tx}.t, \mathsf{tx}.b)$ tuples are unique. While the tie-breaking can be dependent on e.g., transaction ciphertext or metadata, this does not affect our analysis and therefore can be safely ignored for the purpose of our paper.

2.2 Independence of Irrelevant Transactions (IIT)

A useful property for our ordering policy to have is to prevent the ordering decision between transactions tx_a and tx_b to change depending on what other transactions are being ordered; in other words, the ordering decision should not depend on irrelevant transactions. Intuitively, this is done to ensure that an adversary cannot create dummy transactions in order to flip the ordering decision between two transactions, and so that a party's bidding strategy can ignore transactions irrelevant to that party. We define this property of independence of irrelevant transactions (IIT) below.

▶ **Definition 2** (Independence of Irrelevant Transactions). *We say that a policy $\mathbb{P}$ satisfies independence of irrelevant transactions (IIT) if for any pair of transactions $\mathsf{tx}_a, \mathsf{tx}_b$ and any pair of finite subsets $\mathcal{T}_1, \mathcal{T}_2 \subset \mathcal{T}$, the following holds:*

$$\begin{aligned} &\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\} \cup \mathcal{T}_1, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\} \cup \mathcal{T}_1, \mathsf{tx}_b) \\ \Leftrightarrow\ &\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\} \cup \mathcal{T}_2, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\} \cup \mathcal{T}_2, \mathsf{tx}_b). \end{aligned}$$

2.3 IIT Implies a Score-Based Policy

We now show that the IIT property implies that a score-based policy needs to be used – that is, also needs to be independent of the set $\mathcal{T}'$ being ordered.

Intuitively, a score-based policy works as follows: for transaction tx, it assigns a score $S(\mathsf{tx})$ based only on the arrival time $\mathsf{tx}.t$ and the bid $\mathsf{tx}.b$. Here too, scoring ties can be broken in a pre-specified manner. The output sequence is then taken to the descending order of transaction scores. Score-based policies are formally defined below:

▶ **Definition 3** (Score-based policy). *A score is a function $S : \mathcal{T} \to \mathbb{R}$ that assigns to each possible transaction $\mathsf{tx} \in \mathcal{T}$ a score $S(\mathsf{tx})$. An ordering policy $\mathbb{P}$ is called score-based if there exists a score function S such that $\mathbb{P}$ sorts transactions according to S. In other words, there exists S such that for any $\mathcal{T}' \subseteq \mathcal{T}$ and $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}'$, it holds that $\mathbb{P}(\mathcal{T}', \mathsf{tx}_a) < \mathbb{P}(\mathcal{T}', \mathsf{tx}_b)$ if and only if $S(\mathsf{tx}_a) > S(\mathsf{tx}_b)$.*

For finite $\mathcal{T}$, we can directly show that IIT implies score-based policies. To show the result for infinite sets, we need to employ the following set-theoretic axiom (defined below) by Cantor [4]. Similar definitions have also been used in in the context of utility theory [7]

▶ **Property 4** (Cantor's Axiom [4]). *We say that a pair $(\mathbb{P}, \mathcal{T})$ satisfies Cantor's axiom if there exists a countable set $\mathcal{T}' \subseteq \mathcal{T}$ such that for any pair of transactions $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}$ there exists an instance of $\mathbb{P}$ in which some transaction in $\mathcal{T}'$ is ordered between tx_a and tx_b.*

Formally there is a finite set $\mathcal{T}'' \subset \mathcal{T}$ with $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}''$ and a $\mathsf{tx}_c \in \mathcal{T}' \cap \mathcal{T}''$ (possibly $\mathsf{tx}_c = \mathsf{tx}_a$ or $\mathsf{tx}_c = \mathsf{tx}_b$) such that

$$\mathbb{P}(\mathcal{T}'', \mathsf{tx}_a) \leq \mathbb{P}(\mathcal{T}'', \mathsf{tx}_c) \leq \mathbb{P}(\mathcal{T}'', \mathsf{tx}_b),$$

or

$$\mathbb{P}(\mathcal{T}'', \mathsf{tx}_b) \leq \mathbb{P}(\mathcal{T}'', \mathsf{tx}_c) \leq \mathbb{P}(\mathcal{T}'', \mathsf{tx}_a).$$

We can now establish the following correspondence between IIT and score-based policies.

▶ **Theorem 5** (IIT ⇔ Score-Based). *Let $\mathcal{T}$ denote the set of all transactions. The following hold for any ordering policy $\mathbb{P}$:*

1. *If $\mathcal{T}$ is countable, then $\mathbb{P}$ satisfies IIT if and only if it is score-based.*
2. *If $\mathcal{T}$ is uncountable and $(\mathbb{P}, \mathcal{T})$ satisfies Cantor's axiom, then $\mathbb{P}$ satisfies IIT if and only if it is score-based.*

Proof. It is straightforward to see that a score-based algorithm satisfies the independence of irrelevant transactions (since the score of a transaction depends only on itself and not other transactions).

For the opposite direction, we first prove the second part of the theorem (the uncountable case). We define an order $\prec$ over $\mathcal{T}$ where

$$\mathsf{tx}_a \prec \mathsf{tx}_b :\Leftrightarrow \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\}, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\}, \mathsf{tx}_b).$$

Since $\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\})$ is a well-defined for any two transactions $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}$, the order $\prec$ is complete and anti-symmetric. By independence and since $\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b, \mathsf{tx}_c\})$ is a well-defined order for any three transactions $\mathsf{tx}_a, \mathsf{tx}_b, \mathsf{tx}_c \in \mathcal{T}$ we have

$$\begin{aligned}
&\mathsf{tx}_a \prec \mathsf{tx}_b \prec \mathsf{tx}_c\\
\Rightarrow&(\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\}, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\}, \mathsf{tx}_b) \text{ and } \mathbb{P}(\{\mathsf{tx}_b, \mathsf{tx}_c\}, \mathsf{tx}_b) < \mathbb{P}(\{\mathsf{tx}_b, \mathsf{tx}_c\}, \mathsf{tx}_c))\\
\Rightarrow&\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b, \mathsf{tx}_c\}, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b, \mathsf{tx}_c\}, \mathsf{tx}_b) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b, \mathsf{tx}_c\}, \mathsf{tx}_c)\\
\Rightarrow&\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_c\}, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_c\}, \mathsf{tx}_c)\\
\Rightarrow&\mathsf{tx}_a \prec \mathsf{tx}_c
\end{aligned}$$

Therefore, $\prec$ is transitive. We let $\mathsf{tx}_a \preceq \mathsf{tx}_b$ iff $\mathsf{tx}_a \prec \mathsf{tx}_b$ or $\mathsf{tx}_a = \mathsf{tx}_b$.

The Cantor axiom and independence imply that there is a countable $\mathcal{T}' \subset \mathcal{T}$ so that the order $\prec$ satisfies that for any $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}$ there is a $\mathsf{tx}_c \in \mathcal{T}'$ such that

$$\mathsf{tx}_a \prec \mathsf{tx}_b \Rightarrow \mathsf{tx}_a \preceq \mathsf{tx}_c \preceq \mathsf{tx}_b$$

By Theorem 1.1 in [5], this, in turn, implies that there is a numerical representation of the order $\prec$ which is a score $S : \mathcal{T} \to \mathbb{R}$ such that for any two transactions $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}$ we have $\mathsf{tx}_a \prec \mathsf{tx}_b$ if and only if $S(\mathsf{tx}_a) > S(\mathsf{tx}_b)$.

For the first part of the theorem, note that the previous argument also works for a countable $\mathcal{T}$ and in that case we can choose $\mathcal{T}' = \mathcal{T}$ where the Cantor axiom is now trivially satisfied. ◀

▶ Remark 6. The above result extends to the case where the policy creates a weak ordering (which can be made strict through a tie-breaking procedure) rather than a strict ordering of transactions. In that case, Definitions 2 and 3 are adapted to weak orders, and we get a score that might assign the same value to two different transactions. The relaxation to weak orders is useful for the case that the set of transactions is uncountable and not a subset of the real numbers (e.g. if $\mathcal{T} = \mathbb{R}^2_+$). In that case, the Cantor axiom is impossible to satisfy for strict orders but satisfiable for weak orders.

Discussion. We note that in our context, assuming $\mathcal{T}$ is countable or even finite is safe, as there is a finite smallest time increment for timestamps and a finite smallest bid increment. Moreover, the ordering policy deals with ordering transactions in a finite time interval and bids will be upper-bounded by the maximum value in the system (e.g., the maximum number of tokens). However, for the subsequent economic analysis, it will be more convenient to work with the continuum where differences in time stamps and bids can be arbitrarily small.

Having proven that score-based algorithms are essentially the only ones satisfying the independence of irrelevant transactions property, we turn to selecting the most natural one among them. Note that FCFS is the scoring function that corresponds to scoring transactions by their timestamp only while scoring transactions only by bids corresponds to the first-price auction solution. In the next section, we show how our scoring policy TimeBoost corresponds to a simple mixture of these two strategies.

3 TimeBoost Description

We now formally define the TimeBoost ordering policy in this section. As mentioned before, we want TimeBoost to satisfy the independence of irrelevant transactions property (i.e., it needs to be a scoring function based on Theorem 5) and also provide low confirmation-latency for transactions. Therefore, we will only allow TimeBoost to consider transactions within a time g interval; this granularity g can be set suitably based on the particular usecase.

Basic model. Suppose there are n transactions in the g time interval, labeled with $\mathsf{tx}_1, tx_2, \cdots \mathsf{tx}_n$, and sorted by increasing arrival time. Each transaction tx_i is characterized by a pair of a timestamp or arrival time, denoted by t_i, and a bid, denoted by $b_i \geq 0$. Formally, we view a transaction as a tuple of non-negative reals, $\mathsf{tx}_i = (t_i, b_i) \in \mathbb{R}^+ \times \mathbb{R}^{\geq 0}$.

TimeBoost scoring function. Intuitively, for the TimeBoost scoring function, we propose to allow users to "buy time" using their transaction bid; in other words, transactions will be sorted by increasing timestamps (as in FCFS) but now users are allowed to decrease their effective timestamp (i.e., increase their score) through bids.

Formally, the score of a transaction $\mathsf{tx}_i = (t_i, b_i)$ is computed as follows:

$$S(t_i, b_i) = \pi(b_i) - t_i. \tag{1}$$

where $\pi(b_i)$ denotes the priority or advantage gained by bidding b_i. Transactions are now chosen in descending order of their scores.

Choosing a bidding function π. To choose the bidding function π for TimeBoost, we start by defining several natural properties that should be satisfied.

1. $\pi(0) = 0$. This normalization implies that paying 0 bid gives no additional advantage.
2. $\pi'(b) > 0$ for all $b \in \mathbb{R}^+$ where π' denotes the first derivative of π with respect to the bid. This implies that the priority increases with the bid, which gives incentive to bid more for a higher priority.
3. $\lim_{b\to\infty} \pi(b) = g$. This implies that no transaction can outbid a transaction which arrived g time earlier (but any time advantage of less than g can be outbid). Through this, we can guarantee that the transaction ordering can be finalized within time g.
4. $\pi''(b) < 0$ for all $b \in \mathbb{R}^+$ where π'' denotes the second derivative of π with respect to the bid. This means that priority is concave, or equivalently, the cost of producing the (bidding) signal is convex. This is generally necessary to obtain the interior solution of the equilibrium condition.

   The simplest bidding function satisfying the above constraints is the function:

$$\pi(b_i) := \frac{gb_i}{b_i + c} \tag{2}$$

where c is some constant. We will use this as the bidding function for TimeBoost. In the next section, we provide an economic analysis for TimeBoost. For this, we will assume that $c = 1$.

Complexity. For any incoming transaction $\mathsf{tx} = (t, b)$, the sequencer can finalize tx after a delay of $g - \pi(b_i)$. This is because after this point, no later transaction can outbid tx. If transactions arrive at rate r, the space complexity of the sequencing algorithm is $\Theta(r)$ and the computational cost per transaction is $\Theta(\log r)$, assuming pending transactions are stored in a priority queue, ranked by score.

3.1 TimeBoost Economic Analysis Overview

We now describe the model for analyzing the economics for our TimeBoost ordering policy. The next two sections will describe this analysis in detail.

Basic model. Consider an arbitrage opportunity that occurs at some time (w.l.g., this can be taken as time 0). Users (from now on referred to as players) need to now take a decision on (1) how to send their transaction to the sequencer; this corresponds to the investment in latency; and (2) how much extra to bid for their transaction to get higher priority. We will analyze a simple economic model of this decision problem.

Assume that it costs user i the amount $c_i(t)$ to get its transaction received by the sequencer t time after the arbitrage opportunity arises. The only requirement on $c_i(t)$, for now, is that it is decreasing in increasing t. When the arbitrage opportunity arises, a player i has a valuation v_i to have its transaction included for execution the earliest, among those transactions contending for the same opportunity.

Analysis organization. We begin with an analysis with two players in Section 4. Within this, we consider different models based on when the latency investment needs to occur. Broadly, we consider two models for latency investment: ex-ante (Section 4.1) and ex-post (Section 4.2). Ex-ante means that the latency investment needs to happen *before* learning the arbitrage opportunity while ex-post means that the latency investment can occur after learning about the arbitrage opportunity.

In Section 5, we generalize our results to many competing players.

4 Analysis of TimeBoost with 2 Players

As a starting point, assume that there are two players with valuations v_1 and v_2, distributed as per the cumulative distribution functions (CDFs) F_1 and F_2. That is, the probability that the valuation of player i is less or equal to x is equal to $F_i(x)$.

For each valuation v, the player may choose their specific latency investment. We can model this as a function $t_i : V \to \mathbb{R}$, such that, $t_i(v)$ is the latency / time chosen by a player i with valuation v. For simplicity, assume that the cost functions and value distributions are the same: $c_i(t) = c(t)$ and $F_i = F$. Throughout the paper, we assume that $F : [0, 1] \to [0, 1]$ is a uniform distribution with $F_i(x) = x$ iff $x \in [0, 1]$, for $i \in \{1, 2\}$, when final numerical values are derived. Obtaining numerical values for different distribution functions is very similar to that of uniform distribution, but we choose a uniform for simplicity of exposition. However, most of the computations are done for general distribution functions.

We now consider two different assumptions regarding the investment in latency improvement. In the first model (ex-ante), we assume that the players need to invest in their latency infrastructure in advance: they acquire or rent servers close to the sequencer prior to knowing the value of the arbitrage opportunities they are competing for. In the second (ex-post) model, we assume that the players are able to invest in the latency after they learn about their valuation of the arbitrage. This corresponds to the case where the arbitrage opportunity itself takes some time to be realized[3]. In this case, the transaction sender can schedule its transaction through the third-party service, which guarantees the delivery of the transaction

[3] Example of such an opportunity is a 12-second delay on the Ethereum network for a transaction to be scheduled.

within some time interval, once the arbitrage opportunity is realized. In both cases, bidding is naturally assumed to be an interim decision, and in fact, one of the biggest advantages, as the valuation is already learned.

4.1 Ex-Ante Latency Investment

In this model, players learn their valuations only after they have already invested in latency infrastructure. If players can only compete through latency, the interaction between them becomes a static game. We study equilibria solutions of these games. A similar setting is considered in [16]. The results obtained in the following two subsections are concrete cases of folk results in the microeconomic theory; however, we include their proofs for completeness.

4.1.1 Only latency investment

As a simple first step, we start by analyzing the game where only latency investment is allowed. Let x_i be the amount invested in latency by player i (so that he obtains a delay of $t_i(x_i)$). Let V_i denote the valuation random variable of player i. Then, player i has the following ex-ante payoff:

$$\text{Payoff}_i = \begin{cases} E[V_i] - x_i & \text{if player invests strictly more than the other player} \\ \frac{1}{2}E[V_i] - x_i & \text{if he invests an equal amount (assuming random tie-breaking)} \\ -x_i & \text{otherwise} \end{cases}$$

First, we note that there is no pure strategy Nash equilibrium solution of the game, in which player strategy sets consist of $\mathbb{R}^+$. It is easy to show this by case distinction on valuations: there are simple deviating strategies in each case. Next, we focus on the mixed equilibrium solution and obtain the following result.

▶ **Proposition 7.** *There is a symmetric equilibrium in mixed strategies where each player i chooses x_i uniformly at random on the interval $(0, E[V_i])$.*

Proof. By construction, the payoff of player j of playing $x_j \leq E[V_j]$ against the uniform strategy on $(0, E[V_i])$ is

$$F(x_j)E[V_j] - x_j = \frac{x_j}{E[V_i]}E[V_j] - x_j = 0.$$

Choosing a strategy $x_j > E[V_j]$ gives a negative pay-off. Therefore, each $0 \leq x_j \leq E[V_j]$ is the best response of player j, and mixing uniformly among them is also the best response. ◀

The above-described equilibrium is unique up to a change of strategy on a null set and in any mixed equilibrium, both players obtain the same payoffs as in this equilibrium. Note that the result is independent of the latency cost function. The only property used is that if a player invests more than the other player in the latency technology, its transaction is scheduled earlier.

4.1.2 Budget constraints

We now model the fact that players may not have access to an arbitrary amount of money they need to invest to improve their latency, but are instead are constrained by a budget. Let B_i denote the budget of player i, meaning that player i cannot spend more than B_i. We consider an asymmetric case where one (weak) player has a budget $B_1 < E[V_i]$ and the other (strong) player has a larger budget with $B_2 > B_1$. First, note that similar to the previous section with unlimited access to money, there is no pure strategy Nash equilibrium. Therefore, we switch to mixed strategy equilibrium. Let F_i denote the probability distribution of playing different strategies.

▶ **Proposition 8.** *There exists a mixed Nash equilibrium solution in the game in which the weak player receives a payoff of* 0 *and the strong player receives a payoff of* $E[V_i] - B_1$*.*

Proof. The following strategy profile in which the first player plays according to the following (mixed) strategy:

$$F_1(x) = \begin{cases} \frac{x}{E[V_i]} + \frac{E[V_i]-B_1}{E[V_i]}, & x \in (0, B_1], \\ \frac{E[V_i]-B_1}{E[V_i]}, & x = 0, \\ 1, & x > B_1, \end{cases}$$

the second player plays according to

$$F_2(x) = \begin{cases} \frac{x}{E[V_i]}, & x \in [0, B_1), \\ 1, & x \geq B_1, \end{cases}$$

is a mixed strategy equilibrium. The first, weak player obtains an expected payoff 0 for any choice of $0 \leq x_1 < B_1$. The second, strong player obtains an expected payoff of $E[V_i] - B_1$ for any choice $0 < x_2 \leq B_1$. Choosing $x_2 > B_1$ is wasteful for the second player and will not occur in equilibrium. Thus, both players are indifferent between all pure strategies in support of F_1 resp. F_2 and for player 2 choosing an action outside of the support of F_2 is dominated. The mixed strategies form a Nash equilibrium. ◀

Similarly to the unconstrained case, the above-described equilibrium is unique up to a change of strategy on a null set and in any mixed equilibrium, both players obtain the same payoffs as in this equilibrium. Also similarly to the previous section, the result is independent of the latency cost function. The only property to derive this result is that if a player invests more than the other player in the latency technology, its transaction is faster (has a lower timestamp).

4.1.3 Ex-ante Latency with Interim Bidding

We now analyze the model where both latency and bidding are allowed but the latency is ex-ante. That is, investment in latency happens before players learn their valuations but after learning their valuation players can use bidding to improve the transaction score.

We consider a version where players learn the other players' latency investment decisions before bidding. This models the fact that players will typically play the game repeatedly and can therefore observe latency levels of each other.

In the following let $x = (x_1, x_2)$ be the latency investment levels chosen by the two bidders and let $\Delta := t_2(x_2) - t_1(x_1)$ be the corresponding difference in latency. W.l.o.g. assume $\Delta \geq 0$. First, we consider the case that $\Delta = 0$:

▶ **Proposition 9.** *There is a completely separating equilibrium of the bidding game when both bidders have made the same ex-ante investment.*

Proof. Given in Section 4.1.4 ◀

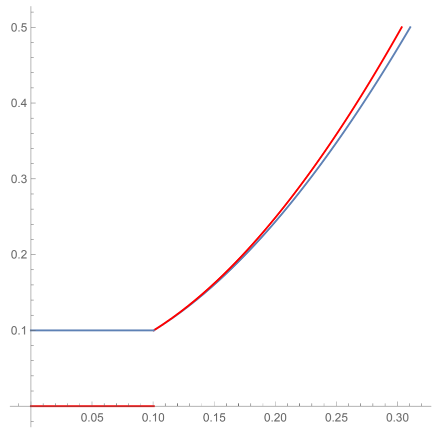


Figure 1 Example of equilibrium signaling functions for $g = 10$ and $\Delta = 0.1$. Timestamps are normalized so that $t_2 = 0$. The blue function is the equilibrium signal $\pi_1(v) - t_1$ for bidder 1 as a function of the valuation. The red function is the equilibrium signal $\pi_2(v) - t_2$ for bidder 2 as a function of the valuation.

Next, we consider the case that $\Delta \neq 0$. For the case of different ex-ante investment we get partially separating equilibria where bidders do not bid for low valuations and bid for high valuations. The bidding strategies are asymmetric in general. However, for sufficiently large g the equilibrium becomes approximately symmetric and approximately efficient. See Figure 1 for a graphical illustration.

▶ **Proposition 10.** *There is an equilibrium of the bidding game which is separating conditional on bidding: There is a threshold* $\sqrt{\frac{\Delta}{g-\Delta}}$*, such that a bidder does not bid if his valuation is below the threshold and bids if his valuation is above the threshold. Conditional on bidding, the high latency bidder i produces a higher signal than the low latency bidder j for equal valuations:* $\pi_i(v) - t_i > \pi_j(v) - t_j$*, for* $v > \sqrt{\frac{\Delta}{g-\Delta}}$.

Proof. Given in Section 4.1.4 ◀

The equilibrium analysis in Propositions 9 and 10 indicates how efficient our transaction ordering policy is as a function of the latency investment of bidders. If bidders have the same latency we have a standard all pay auction which yields a fully efficient outcome. If there is a difference in latency we have no bidding for low valuation bidders and approximately equal signals produced for equal valuations for high valuation bidders. Conditional on entry, low latency bidders underbid and high latency bidders overbid relative to the standard all pay strategies. Efficiency depends on the latency difference and the g parameter. If g is chosen sufficiently large the auction is approximately efficient. A too low g can be detected by low bidding activity. Hence our transaction policy can strike a balance between fairness, low latency and efficiency if properly parameterized.

4.1.4 Proofs

Proof of Proposition 9. We want to determine bidding signals $\pi_1(v_1, \Delta)$ and $\pi_2(v_2, \Delta)$, which are functions of valuations and the difference in latency. For a given Δ denote the inverse of $\pi_1(\cdot, \Delta)$ and $\pi_2(\cdot, \Delta)$ by $\tilde{v}_1(\cdot, \Delta)$ and $\tilde{v}_2(\cdot, \Delta)$. Then bidder 1 solves at the interim stage

$$\max_{\pi \geq 0} Pr[\pi - t_1(x_1) \geq \pi_2(v_2, x) - t_2(x_2)]v_1 - \frac{\pi}{g - \pi} = F(\tilde{v}_2(\pi + \Delta, \Delta))v_1 - \frac{\pi}{g - \pi},$$

We obtain the first order condition:

$$f(\tilde{v}_2(\pi+\Delta,\Delta))v_1\frac{\partial \tilde{v}_2(\pi+\Delta,\Delta)}{\partial \pi} = \frac{g}{(g-\pi)^2}$$

For the uniform distribution, this simplifies to:

$$v_1\frac{\partial \tilde{v}_2(\pi+\Delta,\Delta)}{\partial \pi} = \frac{g}{(g-\pi)^2}.$$

Similarly, for bidder 2 we obtain

$$v_2\frac{\partial \tilde{v}_1(\pi-\Delta,\Delta)}{\partial \pi} = \frac{g}{(g-\pi)^2}.$$

The two equations give a system of differential equations that need to be solved for π_1 and π_2 or alternatively for $\tilde{v}_1$ and $\tilde{v}_2$. Alternatively, we can write the system as:

$$\tilde{v}_1(\pi,\Delta)\frac{\partial \tilde{v}_2(\pi+\Delta,\Delta)}{\partial \pi} = \frac{g}{(g-\pi)^2}. \tag{3}$$

$$\tilde{v}_2(\pi,\Delta)\frac{\partial \tilde{v}_1(\pi-\Delta,\Delta)}{\partial \pi} = \frac{g}{(g-\pi)^2}. \tag{4}$$

The solution to (3) and (4) in case of equal investment (so that $\Delta = 0$) and a symmetric equilibrium is given by the following formula:

$$\tilde{v}_1(\pi,0) = \tilde{v}_2(\pi,0) = \sqrt{2\int_0^\pi \frac{g}{(g-\pi)^2}d\pi} = \sqrt{\frac{2\pi}{g-\pi}}. \tag{5}$$

We solve for the signal as a function of the valuation:

$$v^2 = \frac{2\pi}{g-\pi} \Leftrightarrow \pi = \frac{gv^2}{2+v^2}.$$ ◀

Proof of Proposition 10. When $x_1 \neq x_2$, we can first sum up (3) and (4) to obtain a differential equation for the expected payoff v_1v_2:

$$\frac{d(v_1(\pi)v_2(\pi+\Delta))}{d\pi} = \frac{g}{(g-\pi)^2} + \frac{g}{(g-\pi-\Delta)^2}. \tag{6}$$

Integrating both sides of the differential equation above gives the solution:

$$v_1(\pi)v_2(\pi+\Delta) = \frac{\pi}{(g-\pi)} + \frac{\pi+\Delta}{g-\pi-\Delta} + K. \tag{7}$$

To determine the constant we need to determine boundary conditions. For bidder 1, at the threshold where he is indifferent between bidding and not bidding, we have $\pi_1 = 0$ and for bidder 2, at the threshold where he is indifferent between bidding and not bidding, he needs to overcome the handicap, we have $\pi_2 = \Delta$. At the threshold bidder 2 should make the same profit as from pooling,

$$v_1(0)v_2(\Delta) = \frac{\Delta}{g-\Delta} \Rightarrow K = 0.$$

Combining (7) and (4) we obtain a separable differential equation:

$$\frac{dv_1(\pi,\Delta)}{v_1(\pi,\Delta)} = d\pi \frac{g}{(g-\pi-\Delta)^2}\left(\frac{\pi}{g-\pi}+\frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1}. \tag{8}$$

Combining (7) and (3) we obtain another separable differential equation:

$$\frac{dv_2(\pi+\Delta,\Delta)}{v_2(\pi+\Delta,\Delta)} = d\pi \frac{g}{(g-\pi)^2}\left(\frac{\pi}{g-\pi}+\frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1}. \tag{9}$$

Integrating both parts of the equation (8) solves the (logarithm of) the value as a function of the bid:

$$\ln(v_1(\pi)) - \ln(v_1(0)) = \int_0^{\pi} \frac{g}{(g-\pi-\Delta)^2}\left(\frac{\pi}{g-\pi}+\frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1} d\pi.$$

Similarly, integrating both parts of the equation (9) solves the (logarithm of) the value as a function of the bid:

$$\ln(v_2(\pi+\Delta)) - \ln(v_2(\Delta)) = \int_0^{\pi} \frac{g}{(g-\pi)^2}\left(\frac{\pi}{g-\pi}+\frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1} d\pi.$$

To determine the marginal valuations $v_1(0)$ and $v_2(\Delta)$ at which the two bidders start bidding, note that the support of $\pi_i - t_i$ and that of $\pi_j - t_j$ need to coincide for valuations where we have separation of types. Therefore, $v_1(0) = v_2(\Delta)$. Since $v_1(0)v_2(\Delta) = \frac{\Delta}{g-\Delta}$ it follows that $v_1(0) = v_2(\Delta) = \sqrt{\frac{\Delta}{g-\Delta}}$. This is the threshold where bidders start bidding. It follows that for $\Delta \neq 0$

$$v_1(\pi) = \sqrt{\frac{\Delta}{g-\Delta}}\exp\left(\int_0^{\pi} \frac{g}{(g-\pi-\Delta)^2}\left(\frac{\pi}{g-\pi}+\frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1} d\pi\right)$$

and

$$v_2(\pi) = \sqrt{\frac{\Delta}{g-\Delta}}\exp\left(\int_0^{\pi-\Delta} \frac{g}{(g-\pi)^2}\left(\frac{\pi}{g-\pi}+\frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1} d\pi\right).$$

To compare the equilibrium signals $\pi_1(v) - t_1$ and $\pi_2(v) - t_2$ for $v > \sqrt{\frac{\Delta}{g-\Delta}}$, we need to compare $\pi_1(v) + \Delta$ to $\pi_2(v)$.

From the expressions for the valuations as a function of the bid, we can observe (observe that $\frac{g}{(g-\pi-\Delta)^2} \geq \frac{g}{(g-\Delta)^2}$) that

$$v_1(\pi) > v_2(\pi+\Delta),$$

for $\pi > 0$. It follows that

$$\pi_1(v) \leq \pi_2(v) - \Delta,$$

for $v \geq \sqrt{\frac{\Delta}{g-\Delta}}$. ◀

4.2 Ex-Post Latency with Bidding

We now analyze the ex-post model with bidding; here both the latency investment, and the bid can be made after the valuation is observed. First, we start with only the latency investment decision. The expected utility of player i is equal to:

$$Pr[t(v_i) < t(v_j)]v_i - c(t(v_i)),$$

where $j \in \{1, 2\} \setminus i$.

We can look at this from a dual perspective: by $v(t)$ we define the inverse of $t(v)$. This is the so-called *Revelation Principle.* Instead of some function of the type, we report our type directly. Then, the optimization problem becomes:

$$\max_v Pr[v \geq v_2]v_1 - c(t(v)). \tag{10}$$

By replacing the probability with $F(v)$, we get that it is equivalent to

$$\max_v F(v)v_1 - c(t(v)).$$

By the first order condition, we get:

$$v_1 f(v) - c'(t(v))t'(v)|_{v=v_1} = 0,$$

where f is a density function of the valuation distribution F. By plugging in $v = v_1$, it is equal to:

$$v_1 f(v_1) - c'(t(v_1))t'(v_1). \tag{11}$$

For the uniform distribution and cost function $c = \frac{1}{t}$, first order condition gives the following differential equation:

$$v_1 + \frac{t'(v_1)}{t^2(v_1)} = 0. \tag{12}$$

Solving this equation gives $t(v) = \frac{2}{c_1+v^2}$. By the boundary condition that 0 valuation type should wait infinitely (or equivalently pay 0 in the latency), we obtain the value of the constant in the solution: $c_1 = 0$. Therefore, cost incurred is equal to $\frac{1}{t} = \frac{v^2}{2}$. On average each player pays:

$$\int_0^1 \frac{v^2}{2} f(v)dv|_0^1 = \frac{1}{6},$$

for better latency, before learning their types. The cost of producing score $s = \frac{gm}{m+1} - t$ is:

$$c(s) := m + \frac{1}{t}. \tag{13}$$

We decompose total expenditure into 2 parts, for bidding and for time, by representing m and $c(t(v))$ as functions of v and taking integrals:

$$b(g) := \int_0^1 m(v)f(v)dv \text{ and } \int_0^1 \frac{1}{t(v)} f(v)dv.$$

▶ **Proposition 11.** *The limit of $b(g)$ when g tends to infinity is equal to $\frac{1}{6}$. $b(g)$ is an increasing function in g, for g large enough.*

Proof. Given in Section 4.2.1 ◀

The proposition implies that by taking large enough g, the system extracts almost all value invested in the latency through bidding. Starting from some threshold value on g, extraction increases with increasing g.

We can verify whether the constructed equilibrium is unique by checking the conditions given in [12].

▶ **Example 12.** We can calculate a few values of $b(g)$. In particular, $b(1000) \approx 0.1294$, meaning a player pays approximately 77% of the total expenditure in bids, and $b(10000) \approx 0.1537$, meaning a player pays approximately 92% of the total expenditure in bids.

Note that in the proof of the proposition 11, the total investment in both latency and bidding, $c(v)$, is the same value $\frac{v^2}{2}$, as in the case of only investing in the latency. We show that this is not a coincidence. In general, assume that there is an arbitrary signaling technology described by an increasing, differentiable cost function $C(s)$. The following result shows the revenue equivalence of ex-post bidding:

▶ **Proposition 13.** *Both players spend the same amount on average for any cost function C.*

Proof. Given in Section 4.2.1 ◀

The amount spent depends only on the value belief distribution function.

4.2.1 Proofs

Proof of Proposition 11. The optimization problem of the player in the equilibrium is to minimize cost, subject to the score equation constraint. By plugging in $t = \frac{gm}{m+1} - s$, we obtain the minimization problem:

$$\min_m \left(m + \frac{m+1}{gm - s(m+1)} =: x(m) \right).$$

The first order condition on $x(m)$ gives:

$$\frac{dx(m)}{dm} = 1 + \frac{gm - s(m+1) - (m+1)(g-s)}{(gm - s(m+1))^2} = 1 - \frac{g}{(gm - s(m+1))^2} = 0, \tag{14}$$

gives that the value of m that minimizes the cost function. The solutions of the last equation are $gm - sm - s = \sqrt{g}$ equivalent to $m = \frac{s+\sqrt{g}}{g-s}$ and $gm - sm - s = -\sqrt{g}$ equivalent to $m = \frac{s-\sqrt{g}}{g-s}$, or the boundary condition $m = 0$. For $m = 0$, the value $x(0) = -\frac{1}{s}$, while for $m = \frac{s+\sqrt{g}}{g-s}$, the value

$$x\left(\frac{s+\sqrt{g}}{g-s}\right) = \frac{s+\sqrt{g}}{g-s} + \frac{\frac{s+\sqrt{g}}{g-s} + 1}{g\frac{s+\sqrt{g}}{g-s} - s(\frac{s+\sqrt{g}}{g-s} + 1)} = \frac{1 + 2\sqrt{g} + s}{g - s}.$$

Accordingly, the marginal cost of producing signal s is:

$$c'(s) = \begin{cases} \frac{(1+\sqrt{g})^2}{(g-s)^2}, & \text{if } s > -\sqrt{g}, \\ \frac{1}{s^2}, & \text{if } s \le -\sqrt{g}. \end{cases}$$

We solve a similar differential equation as (11), just with different marginal cost function c', and instead of time function t, we have a score function s of valuation v. The differential equation becomes:

$$vf(v) - c'(s)s'(v) = 0. \tag{15}$$

We need to solve for the $s(v)$ function. For types v with $\frac{2}{v^2} \geq \sqrt{g}$ who only use latency we have the same solution as before

$$s(v) = -\frac{2}{v^2}.$$

The marginal type who is indifferent between using only latency and using a combination of the two technologies is given by

$$u = \sqrt{\frac{2}{\sqrt{g}}}.$$

which gives the boundary condition $s(u) = -\sqrt{g}$ for the differential equation describing the behavior of types who choose a signal $s \geq -\sqrt{g}$:

$$v = \frac{(1+\sqrt{g})^2}{(g-s)^2} s'(v).$$

We obtain the solution

$$s(v) = (4c_1 g^{3/2} + 2c_1 g^2 + 2c_1 g + g(v^2-2) - 4\sqrt{g} - 2)/(2c_1 g + 4c_1\sqrt{g} + 2c_1 + v^2). \tag{16}$$

The value of the constant c is obtained from the boundary condition that a zero-value player does not invest and it is equal to

$$c_1 = \frac{1}{(1+\sqrt{g})^2}.$$

Therefore, plugging in the constant value in the solution (16) and simplifying it gives:

$$s(v) = \frac{gv^2 - 4\sqrt{g} - 2}{v^2+2}.$$

Plugging this into the formula of $c(s)$, gives the cost value as a function of valuation v:

$$c(v) = \frac{1 + 2\sqrt{g} + \frac{gv^2-4\sqrt{g}-2}{v^2+2}}{g - \frac{gv^2-4\sqrt{g}-2}{v^2+2}} = \frac{v^2}{2}.$$

Separate expenditure in the bidding is calculated by the following formula:

$$b(g) = \int_u^1 m(v)f(v)dv = \int_{\sqrt{\frac{2}{\sqrt{g}}}}^1 \frac{\frac{gv^2-4\sqrt{g}-2}{v^2+2} + \sqrt{g}}{g - \frac{gv^2-4\sqrt{g}-2}{v^2+2}} dv =$$
$$\int_{\sqrt{\frac{2}{\sqrt{g}}}}^1 \frac{v^2(g+\sqrt{g}) - 4\sqrt{g} - 2}{2g - 4\sqrt{g} - 2} dv =$$
$$\frac{1}{2g+4\sqrt{g}+2}\left(\frac{g+\sqrt{g}}{3}(1 - \frac{2}{\sqrt{g}}\sqrt{\frac{2}{\sqrt{g}}}) - (4\sqrt{g}+2)(1-\sqrt{\frac{2}{\sqrt{g}}})\right).$$

The dominant term in the nominator above is g and also in the denominator, it is $6g$. Therefore, $\lim_{g\to\infty} b(g) = \frac{1}{6}$. ◀

Proof of Proposition 13. We are interested in the equilibrium signaling strategy $s(v)$. Suppose that this strategy is increasing (so no pooling of types) and differentiable. Then, we can define a differentiable function

$$\tilde{C}(v) := C(s(v)).$$

To figure out what $\tilde{C}(v)$ is, we have to consider an optimization problem with the first player:

$$\max_{v} Pr[v \geq v_2]v_1 - C(s(v)) = Pr[v \geq v_2]v_1 - \tilde{C}(v).$$

Taking first order conditions with respect to v gives:

$$v_1 f(v) - \tilde{C}'(v)|_{v=v_1} = 0,$$

that is,

$$v_1 f(v_1) = \tilde{C}'(v_1).$$

For the uniform distribution:

$$v_1 = \tilde{C}'(v_1).$$

Using the boundary condition $\tilde{C}(0) = 0$ and integrating we get

$$\tilde{C}(v_1) = v_1^2/2.$$

More generally:

$$\tilde{C}(v_1) = \int_{-\infty}^{v_1} vf(v)dv. \qquad ◀$$

5 Analysis of TimeBoost with n players

In this section, we consider n players with the same valuation distribution as in the 2 players case. The optimization problem is now the following:

$$\max_{v} Pr[v \geq \max\{v_2, \cdots, v_n\}]v_1 - c(t(v)),$$

similarly to (10). By replacing the probability with cumulative distribution, this is equivalent to:

$$\max_{v} F_{n-1}(v)v_1 - c(t(v)),$$

where $F_{n-1}(x)$ is a cumulative distribution function of the random variable $X := \max\{X_1, \cdots, X_{n-1}\}$. By independence we have

$$F_{n-1}(x) = F(x)^{n-1}.$$

The first-order condition and plugging in $v = v_1$ gives the following differential equation, similar to (11):

$$f_{n-1}(v_1)v_1 - c'(t(v_1))t'(v_1) = 0,$$

where $f_{n-1}(v_1) = (n-1)v_1^{n-2}$ is a density function of maximum among $n-1$ uniformly distributed random variables. The differential equation w.r.t. $t(v)$ becomes:

$$(n-1)v_1^{n-1} + \frac{t'(v_1)}{t^2(v_1)} = 0.$$

Solving the equation gives $t(v) = \frac{n}{c+(n-1)v^n}$. The same boundary condition ensures that $c = 0$, that is, $t(v) = \frac{n}{(n-1)v^n}$. Each player pays:

$$\frac{n-1}{n}\int_0^1 v^n dv = \frac{n-1}{n}\frac{v^{n+1}}{n+1}|_0^1 = \frac{n-1}{n(n+1)}.$$

Together, the players pay $\frac{n-1}{n+1}$, that converges to 1 as n converges to infinity. Note that the first place in the transaction order is given to the maximum-value player. The average valuation of the maximum value player is $\frac{n}{n+1}$, order statistics. This value also converges to 1 as n tends to infinity.

The analysis is the same as in the case of 2 players, until the differential equation that solves score function s. Instead of (15), for n players we solve:

$$(n-1)vv^{n-1} - c'(s)s'(v) = 0. \tag{17}$$

For types v with $\frac{n}{(n-1)v^2} \geq \sqrt{g}$, who only use latency, we have the same solution as before

$$s(v) = -\frac{n}{(n-1)v^2}.$$

Marginal type investing in bidding is:

$$u = \sqrt{\frac{n}{(n-1)\sqrt{v}}}.$$

Plugging in functional forms of c and s in (17) gives the same limit results as in Proposition 11. Next, we show a revenue equivalence for n players. The argument is similar to 2 players' case. Assume that there is an arbitrary signaling technology described by an increasing, differentiable cost function $C(s)$.

▶ **Proposition 14.** *All n players spend the same amount on average for any cost function C.*

Proof. We are interested in the equilibrium signaling strategy $s(v)$. Suppose that this strategy is increasing (so no pooling of types) and differentiable. Then, we can define a differentiable function

$$\tilde{C}(v) := C(s(v)).$$

To figure out what $\tilde{C}(v)$ is, we have to consider an optimization problem of the first player:

$$\max_v Pr[v \geq \max\{v_2, \cdots, v_n\}]v_1 - \tilde{C}(v) = F(v)^{n-1}v_1 - \tilde{C}(v).$$

Taking first order conditions with respect to v:

$$\left[(n-1)v_1 f(v)F(v)^{n-2} - \tilde{C}'(v)\right]\Big|_{v=v_1} = 0,$$

For the uniform distribution, we get:

$$(n-1)v_1^{n-1} = \tilde{C}'(v_1).$$

Using the boundary condition $\tilde{C}(0) = 0$ and integrating we get

$$\tilde{C}(v_1) = \frac{(n-1)v_1^n}{n}.$$

More generally:

$$\tilde{C}(v_1) = \int_{-\infty}^{v_1} (n-1)vf(v)F(v)^{n-2}dv. \qquad ◀$$

6 Comparison of TimeBoost with a Pure Bidding Policy

We now compare TimeBoost to what to a pure bidding policy. Recall that for the bidding policy, all transactions sent in fixed time intervals of length g are collected, and sorted in decreasing order of their bids. This effectively simulates a first-price all-pay auction for each interval. We note this can be thought of as a quantized version of TimeBoost, because it produces the same sequence that would be produced by first rounding off each transaction's arrival timestamp to the nearest multiple of g and then applying TimeBoost.

Generically speaking, a first-price auction where only the winning bidder pays and first-price all-pay auctions are both *payoff equivalent* for Bayesian-Nash incentive compatible mechanisms, (see e.g., [14]). In our setting, the following result holds for each individual arbitrage opportunity.

▶ **Proposition 15** (see [14]). *The expected payoff of the bidding game where the only the highest bidder pays their bid is equal to the expected payoff in the bidding game where the highest bidder wins but all players pay their bids, independently of valuation distributions.*

For simplicity, to compare TimeBoost with a pure bidding policy, we consider two players. It is straightforward to generalize to more parties. For a given arbitrage opportunity, two cases arise as described below depending on whether transactions can be submitted within the same g-time interval as the arbitrage opportunity or not:

1. Both players can submit their transactions within the same g interval. For the pure bidding policy, if both players can get their transaction submitted inside the same g-time interval as the arbitrage opportunity, then they will both compete for it. It is easy to see that when the valuations of the two parties are the same, the bidding strategy for the pure-bidding policy vs the ex-ante latency with bidding policy will be the same. In other words, in this scenario, TimeBoost maintains the economic properties of the first-price auction pure-bidding policy.
2. Only one player can get its transaction within the same g interval. If only one player can get its transaction inside the same g-time interval as the arbitrage opportunity, then in the pure-bidding policy, that player can pay a 0 bid and still take advantage of it. In contrast, since TimeBoost does not require discrete boundaries, both players will always have g time to submit their transactions (recall that bidding can be used to get priority over any transaction received up to g time earlier). This means that even for a reasonably small g (say 0.5 sec), both parties will always be able to compete for the opportunity. In equilibrium, this results in bids equal to value of the arbitrage.

Analysis for the second case. Suppose the first party (denoted by A) can reach the sequencer in s_1 time, and the second party (denoted by B) can reach in s_2 time, with $s_1 < s_2$. Then, with the pure-bidding policy, A can wait until $g - s_1$ seconds pass since the beginning of a new block creation, and send its transaction to the sequencer at exactly $g - s_1$, while B has to send its transaction by time $g - s_2$ in order to be included in the same block.

Assuming that arbitrage opportunities are uniformly distributed over the g-interval, this means that, with probability $\frac{g-s_1-(g-s_2)}{g} = \frac{s_2-s_1}{g}$, B has no chance to win the race against A, even if it values the arbitrage opportunity much more than A. When g is large (e.g., on Ethereum with 12 sec block times), this latency advantage is not a big issue, as A would only have an advantage with probability $(s_2 - s_1)/12$. In contrast, for faster blockchains, or layer-2 rollups which have shorter block-times to achieve scalability, this latency advantage can be significantly more important in the pure-bidding policy vs in TimeBoost. For instance, when $g = 0.5$sec, A's latency advantage is 24 times greater than what it was in Ethereum. This means that compared to TimeBoost, a pure-bidding strategy will either result in substantial latency competition (when g is small) or will not be able to provide low transaction finalization time (since g will be large).

7 Discussion on Sequencer Decentralization

We now briefly discuss how TimeBoost can be supported by a decentralized sequencer – i.e., a committee of ℓ sequencers (of which at most some f can be dishonest). We only provide possible implementations here; a formal rigorous analysis is outside the scope of this paper.

In the decentralized setting, transactions to be sequenced are now submitted by users to all sequencers instead of just one. Note that as before, threshold decryption techniques can be used for transaction privacy before ordering.

The most natural way to support TimeBoost in a decentralized setting is to have a protocol for sequencers to agree on both the timestamp and the bid of transactions. After this is done, the TimeBoost policy can simply be applied on the consensus output of the decentralized committee to obtain the final ordering. Agreeing on the bid is easy since we can have the same bid be submitted to all sequencers for a given transaction. Agreeing on the timestamp is a more challenging problem since the same transaction can arrive at different nodes. While it adds significant complexity, one potential technique here is to employ a fair-ordering protocol (this can be as a simple as e.g., computing the median timestamp [11, 20] or support more complicated techniques as in [3, 8, 9]). We leave the formal analysis of such a decentralized TimeBoost implementation to future work.

8 Conclusion

We designed TimeBoost: a policy for transaction ordering that takes into account both transaction arrival times and bids. We showed that any ordering scheme that guarantees the independence of different latency races is a generalized scoring rule. By choosing a suitably designed mixture of timestamps and bids, we showed the economic efficiency of the system: transaction senders spend most of their resources on bidding instead of latency improvement, which can later be used by the protocol for improvement and development.

References

1 Kushal Babel and Lucas Baker. Strategic peer selection using transaction value and latency. In *DeFi @ CCS*, pages 9–14, 2022.
2 Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. Clockwork finance: Automated analysis of economic security in smart contracts. In *IEEE S&P*, pages 2499–2516, 2023.
3 Christian Cachin, Jovana Micic, and Nathalie Steinhauer. Quick order fairness. In *FC*, 2022.
4 Georg Cantor. Beiträge zur begründung der transfiniten mengenlehre. *Mathematische Annalen*, pages 481–512, 1895.

5 Christopher P Chambers and Federico Echenique. *Revealed preference theory*, volume 56. Cambridge University Press, 2016.

6 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE S&P*, pages 585–602, 2020.

7 Gerard Debreu. Representation of a preference ordering by a numerical function. *Decision processes*, 3:159–165, 1954.

8 Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *IACR Cryptol. ePrint Arch.*, page 1465, 2021. URL: `https://eprint.iacr.org/2021/1465`.

9 Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for Byzantine consensus. In *CRYPTO*, pages 451–480, 2020.

10 Vijay Krishna. *Auction Theory.* Academic Press, 2002.

11 Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *ACM AFT*, pages 25–36, 2020.

12 Eric Maskin and John Riley. Uniqueness of equilibrium in sealed high-bid auctions. *Games and Economic Behavior*, 45(2):395–409, 2003.

13 Ciamac C. Moallemi and Mehmet Saglam. OR forum - the cost of latency in high-frequency trading. *Oper. Res.*, 61(5):1070–1086, 2013.

14 Roger B. Myerson. Optimal auction design. *Math. Oper. Res.*, 6(1):58–73, 1981.

15 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *IEEE S&P*, pages 198–214, 2022.

16 Ron Siegel. All-pay contests. *Econometrica*, 77:71–92, 2009.

17 Weizhao Tang, Lucianna Kiffer, Giulia Fanti, and Ari Juels. Strategic latency reduction in blockchain peer-to-peer networks. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(2):32:1–32:33, 2023.

18 Anton Wahrstätter, Liyi Zhou, Kaihua Qin, Davor Svetinovic, and Arthur Gervais. Time to bribe: Measuring block construction market, 2023. `arXiv:2305.16468`.

19 Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: Mev countermeasures: Theory and practice, 2022. `arXiv:2212.05111`.

20 Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without Byzantine oligarchy. In *OSDI*, pages 633–649, 2020.

Batching Trades on Automated Market Makers

Andrea Canidio[1]
CoW Protocol, Lisbon, Portugal

Robin Fritsch
Cow Protocol, Lisbon, Portugal
ETH Zürich, Switzerland

Abstract

We consider an automated market maker (AMM) in which all trades are batched and executed at a price equal to the marginal price (i.e., the price of an arbitrarily small trade) after the batch trades. We show that such an AMM is a *function maximizing* AMM (or FM-AMM): for given prices, it trades to reach the highest possible value of a given function. Competition between arbitrageurs guarantees that an FM-AMM always trades at a fair, equilibrium price, and arbitrage profits (also known as LVR) are eliminated. Sandwich attacks are also eliminated because all trades occur at the exogenously-determined equilibrium price. Finally, we show that our results are robust to the case where the batch has exclusive access to the FM-AMM, but can also trade on a traditional constant function AMM.

2012 ACM Subject Classification Applied computing → Economics

Keywords and phrases Arbitrage profits, Loss-vs-Rebalancing (LVR), MEV, Sandwich attacks, AMM, Mechanism design, Batch trading

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.24

Related Version Details missing from the text are available in the full version of the paper.
Full Version: https://arxiv.org/abs/2307.02074 [5]

Acknowledgements We are grateful to Felix Leupold and Martin Köppelmann for initial discussions on batch trading on AMM that led to the writing of this paper. We also thank Haris Angelidakis, Eric Budish, Agostino Capponi, Felix Henneke, Fernando Martinelli, Ciamac Moallemi, Andreas Park, and Anthony Lee Zhang for numerous comments and suggestions.

1 Introduction

This design of Constant Function Automated Market Makers (CFAMMs) has two well-recognized flaws. First, liquidity providers (LPs) trade at a loss whenever there is a rebalancing event. More precisely, when the underlying value of the assets changes, the first informed arbitrageur who trades with the CFAMM earns a profit by aligning the CFAMM price with the new equilibrium price. These profits are at the expense of LPs, who suffer a "loss-vs-rebalancing" (LVR) [16]. Second, traders are routinely exploited by attackers, most commonly via sandwich attacks in which an attacker front-runs a victim's swap with the same swap and then back-runs it with the opposite swap. Doing so allows the attacker to "buy cheap" and "sell expensive" while forcing the victim to trade at less favorable terms.

This paper proposes a novel AMM design that avoids both problems. In our design, all trades that reach the AMM during a period are batched together and executed at a price equal to the new marginal price on the AMM – that is, the price of executing an arbitrarily small trade after the batch trades. We derive the trading function of such an AMM and show two interesting equivalences. First, this AMM is *function maximizing* because, for given

[1] corresponding author

prices, it maximizes the value of a given function subject to a budget constraint. For this reason, we call our design a function-maximizing AMM, or *FM-AMM*. Also, if the function is a standard Cobb-Duglas objective function (i.e., the weighted sum of two natural logs), then for given prices, the FM-AMM LPs run a passive investment strategy: absent trading fees, the total value of the two reserve assets is shared between the two assets according to some pre-specified weights. Finally, we show that an FM-AMM does not satisfy path independence: traders can obtain a better price by splitting their trades into smaller orders, which is why batching is required.

Our main contribution is to consider the behavior of such an AMM in the presence of arbitrageurs, who have private information relative to the equilibrium prices (determined, for example, on some very liquid off-chain location). Competition between arbitrageurs guarantees that the batch always trades at the equilibrium price, and arbitrage profits are eliminated. Intuitively, if this were not the case, some arbitrageurs would want to trade with the batch and, by doing so, would push the price on the batch in line with the equilibrium. This also eliminates all forms of MEV extraction, such as, for example, sandwich attacks: arbitrageurs will always act so to remove deviations from the equilibrium price, therefore making it impossible to manipulate the FM-AMM price. The benefit of contributing liquidity to an FM-AMM relative to a traditional CFAMM is that FM-AMM LPs earn the arbitrage profits generated by rebalancing the CFAMMs. Because these arbitrage profits are larger for more volatile prices (as they lead to more frequent and larger rebalancing, see [16] and [15]), holding everything else equal, the benefit of providing liquidity to an FM-AMM relative to a CFAMM increases with the price volatility.

We extend the theoretical analysis to include a traditional CFAMM, which is important because trading on an FM-AMM may be more expensive than trading on a traditional CFAMM. We introduce the role of the batch operator, who acts in the traders' interest by splitting the batch between the FM-AMM and the traditional CPAMM. We consider the case in which there is a liquid, external trading venue where the equilibrium price is determined and the case in which the entire market is composed of the FM-AMM and the traditional CFAMM. Our results are robust to this extension: we show that arbitrageurs will trade with the batch and on the regular AMM to keep their prices in line with each other (and with the externally-defined equilibrium price, if it exists). The only difference is that the optimal strategy for arbitrageurs may be to trade with the batch and then back-run the batch on the CFAMM. Doing so guarantees that the prices are in equilibrium, and also generates strictly positive profits for the arbitrageur who can successfully back-run the batch.

The remainder of the paper is organized as follows. We now discuss the relevant literature. In Section 2, we introduce the FM-AMM. In Section 3 we consider the behavior of FM-AMM in the equilibrium of a game with informed arbitrageurs and noise traders. Section 4 presents the extension with multiple trading venues. Section 5 discusses the empirical estimation of the returns of providing liquidity to a zero-fee FM-AMM contained in the full version of the paper [5]. The last section concludes. All proofs and mathematical derivations missing from the text are in the appendix.

Relevant literature

The intuition for our main result is closely related to Budish et al. [3], who study the batching of trades in the context of traditional finance as a way to mitigate the high-frequency-trading (HFT) arms race and protect regular (or slow) traders. The main result is that batching trades force informed arbitrageurs to compete in prices instead of speed because the priority of execution within the batch is given based on price. The intuition in our model is similar, although we assume that arbitrageurs compete in quantities rather than in prices: if the price on an FM-AMM differs from the equilibrium price, competing arbitrageurs will submit additional trades to exploit the available arbitrage opportunity, but by doing so, they push the price on the FM-AMM in line with the equilibrium.

Several authors argued that AMM's design allows arbitrageurs to profit at the expense of LPs. Aoyagi [1], Capponi and Jia [6], and Milionis et al. [16] provide theoretical models that illustrate this possibility. In particular, they consider a continuous time model with zero fees and derive a closed-form formula to measure LPs returns and the cost they face when trading with informed arbitrageurs (which they call loss-vs-rebalancing or LVR). Milionis et al. [15] extend this analysis to the case of discrete-time and strictly positive trading fees. They use the term *arbitrageur profits* to indicate LPs losses, a term we adopt because both our model and our empirical analysis are in discrete time and have fees. Aoyagi [1] and Capponi and Jia [6] draw the implication of this cost for liquidity provision.

A second important limitation of CFAMMs is that they enable sandwich attacks (see [17]). These attacks are quantitatively relevant. For example, Torres et al. [21] collected on-chain data from the inception of Ethereum (July 30, 2015) until November 21, 2020 and estimated that sandwich attacks generated 13.9M USD in profits. Qin et al. [18] consider a later period (from the 1st of December 2018 to the 5th of August 2021) and find that sandwich attacks generated 174.34M USD in profits. Our design eliminates these attacks. We are therefore related to the growing literature proposing mechanisms to prevent malicious re-ordering of transactions (of which sandwich attacks are an example), especially those that can be implemented at the smart-contract level [2, 10, 4, 9][2].

Several initial discussions on designing "surplus maximizing" or "surplus capturing" AMMs occurred informally on blog and forum posts (see Leupold [14], Josojo [12], and Della Penna [8]). Goyal et al. [19] provide an axiomatic derivation of the surplus-maximizing AMM. Relative to their work, our contribution is to place this new type of AMM in a context with arbitrageurs and other trading venues. Schlegel and Mamageishvili [20] also study AMM from an axiomatic viewpoint. In particular, they discuss path independence, which FM-AMMs violate.

We conclude by noting that an FM-AMM is also an oracle: it exploits competition between arbitrageurs to reveal on-chain the price at which these arbitrageurs can trade off-chain. It is, therefore, related to the problem of Oracle design (as discussed, for example, by Chainlink [7]).

2 The function-maximizing AMM

In this section, we introduce the main concepts of interest using a simple constant-product function (both for the CFAMM and the FM-AMM), no fees, and keeping formalities to the minimum. The definitions are generalized at the end of the section. We refer the reader to the paper's conclusions for a discussion of additional design choices, and to the main version of the paper [5] for a generalization of our results.

As a preliminary step, we derive the trading function of a constant product AMM, the simplest and most common type of CFAMM. Suppose that there are only two tokens, ETH and DAI. A constant-product AMM (CPAMM) is willing to trade as long as the product of

[2] Another strand of the literature studies how to prevent malicious re-ordering of transactions by modifying the infrastructure that underpins how transactions are sent. See, for example, [13] and the literature review in [11].

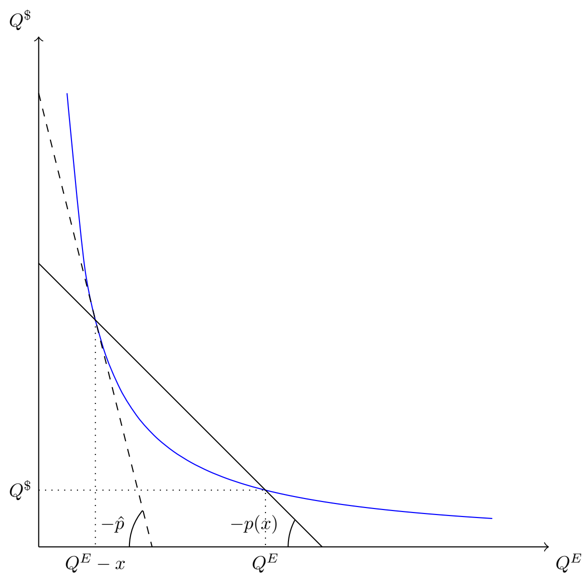


Figure 1 Initially, the liquidity reserves of the CPAMM are $Q^\$$ and Q^E. A trader then purchases x ETH at an average price $p(x)$. Note that, after the trade, the marginal price on the CPAMM (that is, the price for an arbitrarily small trade) is $\hat{p} \neq p(x)$.

its liquidity reserves remains constant (see Figure 1 for an illustration). Call $Q^\$$ and Q^E its initial liquidity reserves in DAI and ETH, respectively, and $p^{CPAMM}(x)$ the average price at which the CPAMM is willing to trade x ETH, where $x > 0$ means that CPAMM is selling ETH while $x < 0$ means that the CPAMM is buying ETH. For the product of the liquidity reserves to be constant, it must be that

$$Q^\$ \cdot Q^E = (Q^\$ + p^{CPAMM}(x)x)(Q^E - x)$$

or

$$p^{CPAMM}(x) = \frac{Q^\$}{Q^E - x}.$$

Note that the marginal price of a CPAMM (i.e., the price to trade an arbitrarily small amount) is given by the ratio of the two liquidity reserves. The key observation is that, in a CPAMM, a trader willing to trade x pays a price that is different from the marginal price after the trade. This is precisely the reason why arbitrageurs can exploit a CPAMM: an arbitrageur who trades with the CPAMM to bring its marginal price in line with some exogenously-determined equilibrium price does so at an advantageous price (and hence makes a profit at the expense of the CPAMM).

Instead, in the introduction, we defined an FM-AMM as an AMM in which, for every trade, the average price equals the marginal price after the trade (we may call this a *clearing-price-consistent AMM*). For ease of comparison with the CPAMM described earlier, suppose that the FM-AMM function is the product of the two liquidity reserves, and hence, that its marginal price is the ratio of its liquidity reserves. The price function $p(x)$ for buying x ETH on the FM-AMM is implicitly defined as

$$p(x) = \frac{Q^\$ + x \cdot p(x)}{Q^E - x},$$

where the RHS of the above expression is the ratio of the two liquidity reserves after the trade. Solving for $p(x)$ yields:

$$p^{FM-AMM}(x) \equiv p(x) = \frac{Q^\$}{Q^E - 2x},$$

which implies that the FM-AMM marginal price is, indeed, the ratio of the liquidity reserves. Hence, a given trade on the FM-AMM generates twice the price impact than the same trade on the traditional CPAMM (cf. the expression for $p^{CPAMM}(x)$).

Interestingly, an FM-AMM can also be seen as a price-taking agent maximizing an objective function. If its objective function is the product of the two liquidity reserves, then for a given price p the FM-AMM supplies x ETH by solving the following problem:

$$x^{FM-AMM}(p) = \text{argmax}_x \left\{ (Q^E - x)(Q^\$ + p \cdot x) \right\}.$$

It is easy to check that the FM-AMM supply function is:

$$x^{FM-AMM}(p) = \frac{1}{2}\left(Q^E - \frac{Q^\$}{p}\right).$$

Hence, to purchase x ETH on the FM-AMM, the price needs to be, again:

$$p^{FM-AMM}(x) = \frac{Q^\$}{Q^E - 2x}.$$

It follows that, whereas a traditional CPAMM always trades along the same curve given by $Q^\$ Q^E$, the FM-AMM trades as to be on the highest possible curve. With some approximation, we can therefore see an FM-AMM as a traditional CPAMM in which additional liquidity is added with each trade. See Figure 2 for an illustration.

A final observation is that the FM-AMM's trading function is equivalent to

$$p \cdot (Q^E - x^{FM-AMM}(p)) = Q^\$ + p \cdot x^{FM-AMM}(p).$$

In other words, for a given p, the values of the two liquidity reserves are equal after the trade. Therefore, the FM-AMM is trading to implement a passive investment strategy, in which the total value of the two reserves is equally split between the two assets (that is, a passive investment strategy with weights 1/2, 1/2). It is easy to check that the FM-AMM can implement any passive investment strategy with fixed weights $(\alpha, 1-\alpha)$ by specifying the objective function as $(Q^E)^\alpha (Q^\$)^{1-\alpha}$ for an appropriate $\alpha \in (0,1)$.

2.1 Path-dependence (or why batching trades is necessary)

CFAMMs (without fees) are path-*independent*: splitting a trade into multiple parts and executing them sequentially does not change the average price of the trade. This property does not hold for an FM-AMM because traders can get better prices by splitting their trade. In fact, they can get approximately the same price as on the corresponding CFAMM by splitting their trade into arbitrarily small parts. This is why an FM-AMM's trading function can be implemented only if trades are batched.

To see this, note that a trade on the FM-AMM with product function changes the reserves as follows:

$$\left(Q^\$, Q^E\right) \quad \to \quad \left(Q^\$\left(\frac{Q^E - x}{Q^E - 2x}\right), Q^E - x\right)$$

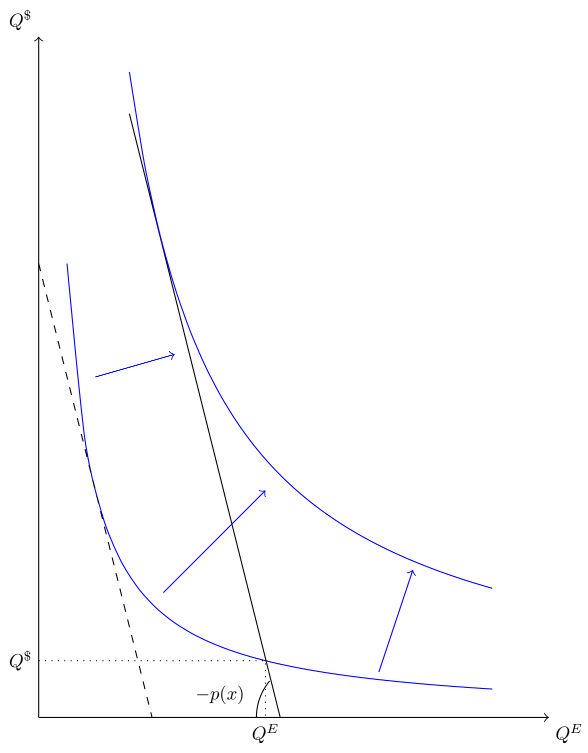


Figure 2 On an FM-AMM, the price at which a given trade x is executed equals the marginal price after the trade is executed. This implies that an FM-AMM "moves up" the curve with each trade.

By instead splitting the trade into smaller parts $\sum_{i=1}^{n} x_i = x$ and executing them sequentially, the reserves of the FM-AMM will change to

$$\left(Q^\$ \prod_{i=1}^{n} \frac{(Q^E - \sum_{j=1}^{i-1} x_j) - x_i}{(Q^E - \sum_{j=1}^{i-1} x_j) - 2x_i}, Q^E - \sum_{i=1}^{n} x_i \right).$$

Setting $x_i = \frac{1}{n}x$ and letting $n \to \infty$ leads to the DAI reserves after the trade being

$$\lim_{n\to\infty} Q^\$ \frac{Q^E - \frac{1}{n}x}{Q^E - \frac{n+1}{n}x} = Q^\$ \frac{Q^E}{Q^E - x}$$

since the product becomes a telescoping product and collapses. This term exactly equals the DAI reserve of a CPAMM after these trades. Hence, to have an FM-AMM, it is necessary to prevent splitting orders by imposing the batching of trades.

2.2 Generalization of definitions

We now generalize our definitions. First of all, an CFAMM is an entity that accepts or rejects trades based on a pre-set rule. Such rule can be derived from the CFAMMs liquidity reserves $(Q^\$, Q^E) \in \mathbb{R}^2_+$ and the CFAMM function $\Psi : \mathbb{R}^2_+ \to \mathbb{R}$. We assume that the CFAMM function is continuous, it is such that $\Psi(Q^\$, 0) = \Psi(0, Q^E) = \Psi(0, 0)$ for all $Q^\$$ Q^E, that it is strictly increasing in both its arguments whenever $Q^\$ > 0$ and $Q^E > 0$, and that it is strictly quasiconcave. The difference between different types of CFAMMs is how the function $\Psi(.,.)$ and the liquidity reserves $(Q^\$, Q^E)$ determine what trades will be accepted and rejected by the CFAMM.

▶ **Definition 1** (Constant Function Automated Market Maker). *For given liquidity reserves* $(Q^\$, Q^E)$ *and function* $\Psi : \mathbb{R}^2_+ \to \mathbb{R}$*, a constant function automated market maker (CFAMM) is willing to trade* x *for* $y = p(x)x$ *if and only if*

$$\Psi(Q^\$ + p(x)x, Q^E - x) = \Psi(Q^\$, Q^E).$$

Our first goal is to define an AMM that is *clearing-price-consistent* in the sense that, for every trade, the average price of the trade equals the marginal price after the trade. This requirement, together with the specification of a marginal price, already fully defines the AMM. In particular, a clearing-price-consistent AMM can be defined to match the marginal price for an arbitrary CFAMM function Ψ.

▶ **Definition 2** (Clearing-Price Consistent AMM). *For given liquidity reserves* $(Q^\$, Q^E)$ *and function* $\Psi : \mathbb{R}^2_+ \to \mathbb{R}$*, let*

$$p_\Psi^{margin}(Q^\$, Q^E) = \frac{\frac{\partial \Psi(Q^\$, Q^E)}{\partial Q^E}}{\frac{\partial \Psi(Q^\$, Q^E)}{\partial Q^\$}}$$

be the marginal price of the AMM for reserves $Q^\$, Q^E$*. A clearing-price consistent AMM is willing to trade* x *for* $y = p(x)x$ *if and only if*

$$p(x) = p_\Psi^{margin}\left(Q^\$ + p(x)x, Q^E - x\right). \tag{1}$$

Note that, given our assumptions on $\Psi(.,.)$, whenever $Q^\$ > 0$ and $Q^E > 0$, the marginal price is strictly increasing in the first argument and strictly decreasing in the second argument, converges to zero as $Q^E \to \infty$ or $Q^\$ \to 0$, and to infinity as $Q^E \to 0$ or $Q^\$ \to \infty$.

Second, we define a *function-maximizing AMM (FM-AMM)* that maximizes the objective function Ψ instead of keeping it constant:

▶ **Definition 3** (Function-Maximizing AMM). *For given liquidity reserves* $(Q^\$, Q^E)$ *and function* $\Psi : \mathbb{R}^2_+ \to \mathbb{R}$*, a function-maximizing AMM is willing to trade* x *for* $y = p(x) \cdot x$ *if and only if* $p(x) = x^{-1}(p)$*, where*

$$x(p) := argmax_x \left\{\Psi\left(Q^\$ + p \cdot x, Q^E - x\right)\right\}. \tag{2}$$

The next proposition establishes the equivalence between clearing-price-consistent and function-maximizing AMMs.

▶ **Proposition 4.** *For given liquidity reserves* $(Q^\$, Q^E)$ *and function* $\Psi : \mathbb{R}^2_+ \to \mathbb{R}$*, an AMM is function maximizing if and only if it is clearing-price consistent.*

Proof. Under our assumptions, solving (2) is equivalent to satisfying the first-order condition, which is equivalent to (1). ◀

3 The model

Equipped with the full description of an FM-AMM, we can now study its behavior in an environment with traders and arbitrageurs. We limit our analysis to the product function.

The game comprises n noise traders, each wanting to trade a_i units of ETH. We adopt the convention that if $a_i > 0$ trader i wants to buy ETH, while if $a_i < 0$ trader i wants to sell ETH. Call $A = \sum_i a_i$ the aggregate demand for ETH from noise traders, assumed small relative to the FM-AMM liquidity reserves Q^E and $Q^\$$.[3] Next to traders, a large number of cash-abundant, competing arbitrageurs, who can trade as part of the batch and on some external trading venue, assumed much larger and more liquid than the combination of our n traders and the FM-AMM. The equilibrium price for ETH on this external trading venue is p^* and is unaffected by trades on the FM-AMM. Arbitrageurs aim to profit from price differences between the FM-AMM and the external trading venues. Arbitrage opportunities will be intertemporal (over short intervals). Hence, for ease of derivations, we assume that arbitrageurs do not discount the future.

The timing of the game is discrete. Even periods are when on-chain transactions occur. Even periods are, therefore, better interpreted as different blocks. Odd periods are when off-chain events occur. In these periods, first, the equilibrium price p^* may change, and then traders/arbitrageurs can submit trades for inclusion in the batch.

We are now ready to derive our main proposition.

▶ **Proposition 5.** *Suppose that, at the end of an even period, the reserves of the FM-AMM are Q^E and $Q^\$$. In the equilibrium of the subsequent odd period, after p^* is realized, noise traders will collectively submit trade A to the batch, and arbitrageurs will collectively submit trade $y(p^*)$ such that $p^{FM-AMM}(A + y(p^*)) = p^*$. Hence, in equilibrium, the FM-AMM price is always equal to p^*.*

The key intuition is that all arbitrageurs can submit trades on the batch. Hence, there is no equilibrium in which there is an exploitable arbitrage opportunity; otherwise, some arbitrageurs will want to submit additional trades on the batch. Arbitrage opportunities are absent whenever $p^{FM-AMM}(A + y(p^*)) = p^*$, which is the unique equilibrium. Hence, competition between arbitrageurs guarantees that the price at which the AMM trades is always correct, even if the AMM has no way of directly observing such price. Unlike a traditional CPAMM, arbitrageurs earn zero by rebalancing an FM-AMM.

4 Extension: multiple trading venues

We modify our theoretical model by introducing a traditional CPAMM. This is a relevant extension because, as already discussed, trading on FM-AMM may be more expensive than trading on a traditional CPAMM, in the sense that for given liquidity pools, a given trade on a CF-AMM has twice the price impact than the same trade on a CPAMM. Naive noise traders may therefore ignore the equilibrium effects and prefer to trade on the CPAMM.

To address this concern, we introduce the figure of the batch operator as the sole agent who can access the FM-AMM. The batch operator has two roles:[4]

- It enforces the batching of trades, including uniform prices for all trades on the batch.
- Conditional on batching trades, the batch operator acts in the traders' interest. In particular, the batch operator will optimally split a trade between the FM-AMM and the traditional CPAMM to obtain the best possible price for the traders in the batch

[3] In practice, we assume that trading A on the FM-AMM has a negligible price impact. Else, we should treat orders from noise traders as limit orders. In this case, all our results continue to hold at the cost of additional notation.

[4] The batch operator is modeled around CoW Protocol (`www.cow.fi`). CoW Protocol collects intentions to trade off-chain, which are executed as a batch by accessing multiple trading venues. Cow Protocol enforces uniform clearing prices so that all traders in the same batch face the same prices.

The rest of the game is similar to the one discussed earlier, except that the large trading venue in which the equilibrium price is determined may or may not exist. If it exists, we say that the market is deep. Else, the only trading venues are the FM-AMM and the CPAMM, and we say that the market is shallow.[5]

Again, there are noise traders and arbitrageurs. Arbitrageurs trade so to exploit price differences between the trading venues that they can access. Even periods of the game are when on-chain transactions occur and are better interpreted as different blocks. Odd periods are when off-chain events occur. In these periods, traders/arbitrageurs can submit trades for inclusion in a batch. Other events may also happen, depending on whether the market for trading ETH against DAI is deep or shallow. For simplicity, both FM-AMM and the CPAMM charge zero fees.

We start by deriving the optimal behavior of the batch operators

4.1 The batch operator optimal behavior

The first step is to derive the optimal behavior of the batch operator, who can trade both on an FM-AMM (with reserves $Q^{\$,FM-AMM}$ and $Q^{E,FM-AMM}$) and on a CPAMM (with reserves $Q^{\$,CPAMM}$ and $Q^{E,CPAMM}$). If the batch needs to purchase X, it will split the trade between x^{CPAMM} and $x^{FM-AMM} = X - x^{CPAMM}$ so as to minimize the total expenditure, that is

$$\min_{x^{CPAMM}} \left\{ x^{CPAMM} \frac{Q^{\$,CPAMM}}{Q^{E,CPAMM} - x^{CPAMM}} + x^{FM-AMM} \frac{Q^{\$,FM-AMM}}{Q^{E,FM-AMM} - 2x^{FM-AMM}} \right\}$$

There is a special case in which the math simplifies: that of equal reserves in the two AMMs. In this case, the solution to the above minimization problem is $x^{CPAMM} = \frac{2}{3}X$; because the price impact on the FM-AMM is twice that of the traditional CPAMM, the batch will optimally route a given trade 1/3 on the FM-AMM and 2/3 on the standard CPAMM. As a result, the (average) price that the batch pays on the CPAMM equals the price the batch pays on the FM-AMM. We illustrate this case in Figure 3.

To simplify the analysis, we, therefore, assume that, initially, $Q^{\$,CPAMM} = Q^{\$,FM-AMM}$ and $Q^{E,CPAMM} = Q^{E,FM-AMM}$.

4.2 Deep market

We say that the market is deep whenever there is an exogenously-determined equilibrium price for ETH, which we call p^*. In this case, during odd periods (off-chain), first, the equilibrium price p^* may change, and then traders/arbitrageurs can submit trades for inclusion in the batch.

Suppose that, initially, the equilibrium price is p_0^*, and both AMMs are in equilibrium so that $p_0^* = \frac{Q^\$}{Q^E}$. In a given odd period, the equilibrium price then jumps to $p_1^* \neq p_0^*$, and stays constant afterward. We study the adjustment to the new equilibrium depending on whether the batch or an arbitrageur is first in the block.[6]

[5] Of course, the ETH/DAI market is deep. The point is to generalize the example to token pairs that may not be.

[6] In deriving the equilibrium, we assume that a player observes on-chain transactions included earlier in a block and can use this observation to craft their transaction. This is a shorthand for a transaction that specifies a strategy to follow as a function of the possible on-chain behavior of the preceding players.

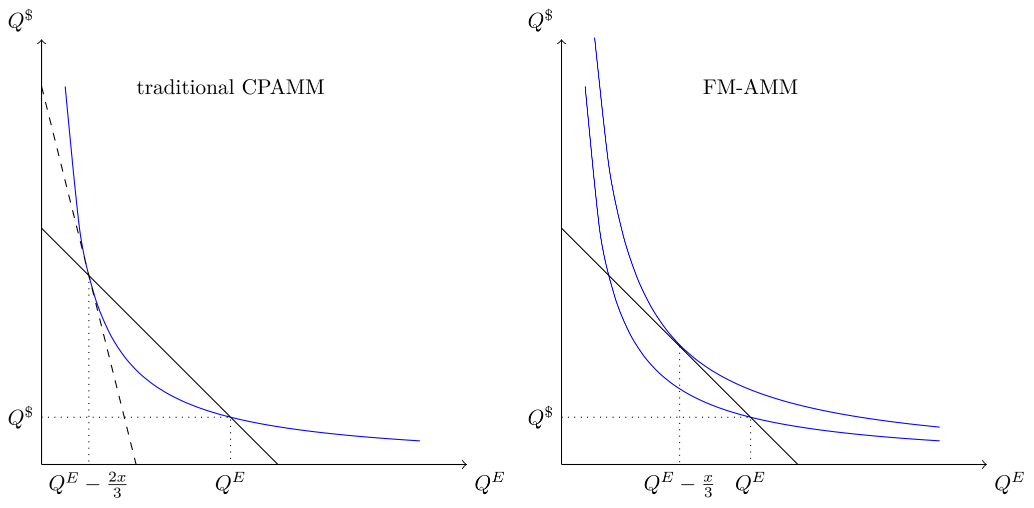


Figure 3 Starting from the initial reserves (assumed equal in the two AMMs) the batch trades so that the new marginal price on the FM-AMM equals the average price on the traditional CPAMM (represented by the two solid black lines). Note that the marginal price on the traditional CPAMM (dashed line) differs from the average price at which the batch trades.

The batch is at the top of the block

In this case, upon observing the new price, arbitrageurs submit trades for inclusion in the batch. Competition among arbitrageurs guarantees that their total trade on the batch is such that the price on the batch is p_1^*. This implies that the price on the FM-AMM is exactly p_1^*. However, if the batch traded on the traditional CPAMM at an *average* price of p_1^*, then the new *marginal* price on the traditional CPAMM will be different from p_1^*. More precisely, call $x^{CPAMM,*}$ the amount of ETH exchanged by the batch on the traditional CPAMM, defined as

$$x^{CPAMM,*}: \; p_1^* = \frac{Q^\$}{Q^E - x^{CPAMM,*}}$$

After such trade, the marginal price on the traditional CPAMM is given by:

$$\frac{Q^\$ + p_1^* x^{CPAMM,*}}{Q^E - x^{CPAMM,*}} = \frac{p_1^* Q^E}{\frac{Q^\$}{p_1^*}} = \frac{(p_1^*)^2}{p_0^*} \neq p_1^*$$

where $Q^\$ + p_1^* x^{CPAMM,*} = p_1^* Q^E$ and $Q^E - x^{CPAMM,*} = \frac{Q^\$}{p_1^*}$ are, respectively, the DAI and ETH reserves on the CPAMM after the batch settles its trade. Hence, if $p_1^* > p_0^*$, then the marginal price on the traditional CPAMM will be greater than p_1^*; if instead $p_1^* < p_0^*$, then the marginal price on the traditional CPAMM will be smaller than p_1^*. The arbitrageur trading after the batch will then perform a trade y given by:[7]

$$\max_y \left\{ y \left(p_1^* - \frac{p_1^* Q^E}{\frac{Q^\$}{p_1^*} - y} \right) \right\}$$

[7] Note that if $y > 0$, then the arbitrageur buys from the CPAMM to sell at the equilibrium price p_1^*. If instead $y < 0$, then the arbitrageur buys at the equilibrium price p_1^* to sell to the CPAMM.

with first-order conditions

$$\frac{p_1^* + \left(\frac{p_1^* Q^E}{\frac{Q^\$}{p_1^*} - y}\right) y}{\frac{Q^\$}{p_1^*} - y} = p_1^*$$

It is easy to check that the numerator of the above expression is the DAI reserves after the batch settles and after the arbitrageur trades y. Similarly, the denominator is the ETH reserves after the trades from the batch and the arbitrageurs. Therefore, the arbitrageur's optimal trade re-aligns the marginal price on the CPAMM with the equilibrium price p_1^*.

An arbitrageur is at the top of a block

To start, note that, independently of what trades were submitted to the batch, the arbitrageur at the top of the block will exploit the arbitrage opportunity available on the traditional CPAMM, and trade on the CPAMM until its marginal price equals p_1^*. Hence, when arbitrageurs submit trades on the batch, they anticipate that the CPAMM price will be p_1^* by the time the batch trades. Competition guarantees that these arbitrageurs submit trades y to the batch until its price is also p_1^*, that is:

$$y: \quad p_1^* = \frac{Q^\$}{Q^E - 2(A + y)}$$

In this case, therefore, the batch does not trade on the traditional CPAMM, but only on the FM-AMM.

4.3 Shallow market

The market is shallow whenever the only venues to trade ETH against DAI are the FM-AMM and the traditional CPAMM. Hence, there is no externally-determined equilibrium price. The only arbitrage possibility is to exploit price differences between the FM-AMM and the traditional CPAMM.

Again, suppose that the two AMMs have equal reserves and, therefore, equal marginal prices. The source of the exogenous variation is the arrival of an aggregate trade A on the batch. After observing A, arbitrageurs can submit their trades to the batch. Again, there are two cases to consider, depending on who trades first.

The batch is at the top of the block

To start, note that if arbitrageurs were absent, then after the batch settles A (1/3 on the FM-AMM, the rest on the traditional CPAMM) the marginal prices of the two exchanges would be misaligned. There is therefore an arbitrage opportunity that the first arbitrageur can exploit, as the next proposition shows.

▶ **Proposition 6.** *In equilibrium, the first arbitrageur trading after the batch purchases*

$$y^* = 3\sqrt{2Q^E(2Q^E - A)} - 2(3Q^E - A) \tag{3}$$

with the batch and then sells the same amount on the CPAMM. The other arbitrageurs do not trade.

The proof of the proposition shows that y^* has the same sign as A. Hence, the first arbitrageur trades in the same direction as noise traders to move the price on the CPAMM even more, and then performs the opposite trade on the CPAMM. Furthermore, he can preempt the other arbitrageurs from exploiting the arbitrage opportunity. By doing so, he earns strictly positive profits.

The arbitrageur is at the top of the block

Also here, competition among arbitrageurs guarantees that, collectively, they trade to align the price of ETH on the batch with the marginal price of ETH on the CPAMM. However, unlike the previous case, here, both the first arbitrageur in the block and the first arbitrageur after the batch trade.

▶ **Proposition 7.** *In equilibrium, the first arbitrageur after the batch buys with the batch and then sells on the CPAMM. Again, his optimal trade is y^* given by* (3)*. Furthermore, the first arbitrageur purchases*

$$\sqrt{(Q^E)^2 - Q^E\frac{2}{3}(A+y^*)} - Q^E$$

on the CPAMM and then sells the same amount on the batch. The other arbitrageurs do not trade.

We previously discussed that y^* and A have the same sign. Hence, if for example $A > 0$, then the first arbitrageur purchases on the CPAMM (cheaply) and then sells on the batch (more expensive) to bring the CPAMM price in line with the price on the FM-AMM. If instead $A < 0$, the first arbitrageur sells on the CPAMM and then buys on the FM-AMM. The batch then trades both on the FM-AMM and the CPAMM, therefore moving the price on the CPAMM. Finally, similarly to the previous case, the first arbitrageur after the batch exploits the arbitrage opportunity by back-running the batch: he purchases with the batch and immediately sells on the CPAMM so to align the price on the FM-AMM with the marginal price on the CPAMM.

5 Discussion of empirical results

In the full version of the paper [5], we perform an empirical analysis to quantify the return of providing liquidity to a zero-fee FM-AMM, and compare them with the empirical returns of providing liquidity on Uniswap V3. We now briefly describe our method and report our main results.[8]

To simulate FM-AMM returns, we consider a counterfactual in which an FM-AMM existed from October 2022 to March 2023. We retrieve price data from Binance for several trading pairs and use the Proposition 5 to simulate an FM-AMM pool rebalanced to the Binance price in regular intervals (12 seconds, or 1 block). These rebalancing trades determine the evolution of the FM-AMM reserves and thereby the returns of its liquidity providers. Importantly, because we consider a zero-fee FM-AMM, its LPs earn no fees from noise traders. Equivalently, we could also assume that the FM-AMM does not receive any noise

[8] Note that in the full version of the paper, we consider how the return of providing liquidity of an FM-AMM changes with the fees charged by FM-AMM and with the frequency of rebalancing of the FM-AMM. Here we only report the results for a zero-fee FM-AMM that rebalances every block.

trading volume and is only rebalanced by arbitrageurs. Hence, the estimated LP returns should be considered a lower bound to the possible returns generated by an FM-AMM that also earns revenues from noise traders.

We then compare the returns of providing liquidity to our simulated FM-AMM to the empirical returns of providing liquidity to the corresponding Uniswap v3 pool. To calculate the return on a liquidity position in a Uniswap v3 pool, we consider three pools for which we also have Binance price data: WETH-USDT (with fee 0.05%), WBTC-USDT (with fee 0.3%), and WBTC-WETH (with fee 0.05%).[9] In each case, we simulate the return of a small full-range position. We then query the amount of fees the pool earned in a given block and the amount of liquidity that is "in range" at the end of the same block. [10] We then use the size of the simulated liquidity position in range to calculate the fees it earns.

Our method is based on two assumptions. First, we assume that the simulated liquidity position is too small to affect price slippage, the volume of trades, and the incentive to provide liquidity by other LPs. We also implicitly assume that the liquidity in the range is constant during a block. This last assumption introduces some non-systematic inaccuracies in our estimation. For example, if within a block, first some fees are collected and then some additional liquidity is introduced, our method attributes a fraction of these fees to the new liquidity even if, in reality, it did not earn any. If, instead, first some fees are collected and then some liquidity is withdrawn, our method does not attribute any of the fees to the liquidity that was withdrawn, while in reality, it did earn some fees. Similarly, if a price changes tick, our method shares all fees collected in a block among the liquidity available in the last tick, whereas, in reality, some of the fees are shared among the liquidity available at the initial tick.[11]

Note that our results do not depend on the size of the initial liquidity position. On Uniswap v3, a larger initial position earns proportionally more fees, but its ROI is the same. Similarly, on an FM-AMM, the size of the rebalancing trade scales proportionally with the available liquidity so that, again, its ROI is independent of its initial size. Also, for both Uniswap v3 and the FM-AMM, we consider a liquidity position that is non-concentrated (i.e. , a position over the entire price range $[0, \infty]$). If both positions are concentrated in the same (symmetrical) way, both Uniswap v3 fees and FM-AMM returns increase by the same factor as long as the price does not go out of range. So the comparison does not change, and the full-range comparison already constitutes a general comparison.

Our results are mixed: whether providing liquidity to our simulated FM-AMM generates higher returns than providing the same liquidity to Uniswap v3 depends on the token pair and the period we consider. However, these returns are similar: at the end of the 6-months period we consider, the differences in returns between an FM-AMM and Uniswap v3 across the three pools we study are -0.25% (for the ETH-USDT pool), 0.03% (for the BTC-USDT pool) and 0.11% (for the ETH-BTC pool). Also, during the period we consider, the maximum difference in value between the two liquidity positions (relative to their initial values) is 0.33% (for the ETH-USDT pool), 0.15% (for the BTC-USDT pool), and 0.12% (for the ETH-BTC pool). We conclude that the lowest bound on the return to providing liquidity to an FM-AMM is similar to the empirical return to providing liquidity to Uniswap v3.

[9] Note that, whereas most Binance prices are expressed in $USDT$, this stablecoin is not widely used in Uniswap v3: at the time of writing, if we exclude stablecoin-to-stablecoin pairs, the two pools we consider are the only pools with USDT in the top 30 Uniswap v3 pools.

[10] We use the Uniswap v3 subgraph to query the data. `https://thegraph.com/hosted-service/subgraph/uniswap/uniswap-v3`

[11] Besides being non-systematic, the inaccuracies introduced are likely to be extremely small. For the ETH-USDT pool, we calculate that the difference between assuming liquidity to be constant over *two* blocks instead of over one block is 0.0002% over 6 months. We expect the inaccuracies from assuming the liquidity to be constant over one block to be the same magnitude.

6 Conclusion

We conclude by discussing several design choices that are relevant to the implementation of an FM-AMM.

The first design choice is how to enforce the batching of trades. In the model, we assumed that the FM-AMM enforces batching by collecting intentions to trade off-chain and settling them on-chain at regular intervals, with all trades settled simultaneously facing the same prices. However, there could be other ways to enforce batching, for example, by leveraging proposer-builder separation (or PBS)[12]. In PBS, block builders (entities that assemble transactions in a block that are then forwarded to a proposer for inclusion in the blockchain) could compute the net trades that will reach the FM-AMM during that block. Builders will then include a message announcing this value at the beginning of the block, which the FM-AMM uses to compute the price at which all trades will be executed. If the proposer's announcement turns out to be correct at the end of the block, the FM-AMM will reward the builder (punishments can also be introduced if the block builder report is incorrect, see [14]).

A second design choice is the fee structure. An FM-AMM can charge fees in at least two ways: a fee could be charged for including a trade on the batch, and a separate one could be charged on the net trade that the batch settles on FM-AMM. The difference between the two fees is that some of the trades may be netted already on the batch without ever reaching the FM-AMM. See the full version [5] for an in-depth analysis of this problem.

Another important design choice for any AMM, that is omitted in this work, is what function to use, particularly its curvature: the degree to which the price changes with a given trade. Capponi and Jia [6] show that, in a traditional CPAMM, the choice of curvature is determined by a tradeoff. When the equilibrium price changes and the curvature is high, the CPAMM adjusts its price more rapidly, and LVR is low. At the same time, the amount LPs earn as trading fees is also low. On the other hand, if the curvature is low and the equilibrium price changes, then LVR is high. But because the CPAMM trades more, the amount earned by LPs as fees is also high. The optimal curvature balances these two elements and depends on the trade volume from noise traders vs. arbitrageurs. The fact that the FM-AMM always trades at the equilibrium price suggests that it should encourage trading by adopting a flatter curve relative to a traditional CPAMM. Formalizing this intuition is left for future work.

A final observation is that an FM-AMM eliminates adverse selection by creating competition among multiple informed arbitrageurs. However, if there is a single, informed trader, then this trader can still enjoy information rents and trade at an advantage against FM-AMM LPs.

References

1 Jun Aoyagi. Liquidity provision by automated market makers. *working paper*, 2020. URL: https://dx.doi.org/10.2139/ssrn.3674178.

2 Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. Enter the hydra: Towards principled bug bounties and {Exploit-Resistant} smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1335–1352, 2018.

3 Eric Budish, Peter Cramton, and John Shim. The high-frequency trading arms race: Frequent batch auctions as a market design response. *The Quarterly Journal of Economics*, 130(4):1547–1621, 2015.

[12] https://ethereum.org/en/roadmap/pbs/

4 Andrea Canidio and Vincent Danos. Commitment against front running attacks, 2023. arXiv:2301.13785.
5 Andrea Canidio and Robin Fritsch. Arbitrageurs' profits, lvr, and sandwich attacks: batch trading as an amm design response, 2023. arXiv:2307.02074.
6 Agostino Capponi and Ruizhe Jia. The adoption of blockchain-based decentralized exchanges, 2021. arXiv:2103.08842.
7 Chainlink. What is the blockchain oracle problem? Retrieved from https://chain.link/education-hub/oracle-problem on May 24, 2023, 2020. Online forum post.
8 Nicolás Della Penna. Mev minimizing amm (minmev amm). Retrieved from https://ethresear.ch/t/mev-minimizing-amm-minmev-amm/13775 on May 24, 2023, september 1 2022. Online forum post.
9 Matheus V. X. Ferreira and David C. Parkes. Credible decentralized exchange design via verifiable sequencing rules, 2023. arXiv:2209.15569.
10 Joshua S Gans and Richard T Holden. A solomonic solution to ownership disputes: An application to blockchain front-running. Technical report, National Bureau of Economic Research, 2022.
11 Lioba Heimbach and Roger Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, AFT '22, pages 47–60, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3558535.3559784.
12 Josojo. Mev capturing amm (mcamm). Retrieved from https://ethresear.ch/t/mev-capturing-amm-mcamm/13336 on May 24, 2023, august 4 2022. Online forum post.
13 Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. Cryptology ePrint Archive, Paper 2020/269, 2020. URL: https://eprint.iacr.org/2020/269.
14 F. Leupold. Cow native amms (aka surplus capturing amms with single price clearing). Retrieved from https://forum.cow.fi/t/cow-native-amms-aka-surplus-capturing-amms-with-single-price-clearing/1219/1 on May 24, 2023, november 1 2022. Online forum post.
15 Jason Milionis, Ciamac C. Moallemi, and Tim Roughgarden. Automated market making and arbitrage profits in the presence of fees, 2023. arXiv:2305.14604.
16 Jason Milionis, Ciamac C. Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Automated market making and loss-versus-rebalancing, 2023. arXiv:2208.06046.
17 Andreas Park. Conceptual flaws of decentralized automated market making. Technical report, Working paper, University of Toronto, 2022.
18 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022.
19 Geoffrey Ramseyer, Mohak Goyal, Ashish Goel, and David Mazières. Augmenting batch exchanges with constant function market makers, 2023. arXiv:2210.04929.
20 Jan Christoph Schlegel, Mateusz Kwaśnicki, and Akaki Mamageishvili. Axioms for constant function market makers, 2023. arXiv:2210.00048.
21 Christof Ferreira Torres, Ramiro Camino, et al. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1343–1359, 2021.

A Mathematical derivations

Proof of Proposition 5. The fact that $y(p^*)$ is the unique equilibrium is easily established by contradiction: suppose the equilibrium is $y' \neq y(p^*)$, which implies $p^{FM-AMM}(A+y') \neq p^*$. Then by the fact that $p^{FM-AMM}(A+y')$ is locally continuous, an arbitrageur could submit an additional trade z such that $p^{FM-AMM}(A+y'+z) \neq p^*$ and earn strictly positive profits, which implies that y' is not an equilibrium. It is also easy to establish that $y(p^*)$ is an equilibrium, as no arbitrageur has any incentive to deviate. ◀

Proof of Proposition 6. Suppose that arbitrageurs submit an aggregate trade y on the batch and later $-y$ on the traditional CPAMM. The price on the batch is:

$$\frac{Q^\$}{Q^E - \frac{2}{3}(A+y)}$$

and the price on the reverse trade is

$$\frac{Q^\$ Q^E}{(Q^E - \frac{2}{3}(A+y))(Q^E - \frac{2}{3}(A+y)+y)}.$$

Competition between arbitrageurs guarantees that the marginal arbitrageur earns zero profits. This, in turn, implies that the price on the FM-AMM equals the marginal price on the CPAMM, that is

$$\frac{Q^\$}{Q^E - \frac{2}{3}(A+y)} = \frac{Q^\$ Q^E}{(Q^E - \frac{2}{3}(A+y)+y)^2}$$

with a unique solution in the range $[2A - 3Q^E, \frac{3}{2}Q^E - A]$ (so that both prices are strictly positive and well-defined), given by

$$y^* = 3\sqrt{2Q^E(2Q^E - A)} - 2(3Q^E - A).$$

Note that y^* is strictly increasing in A and is equal to zero whenever $A = 0$. This can be seen by verifying that the derivative with respect to A is positive for $A < \frac{7}{8}Q_E$ (which is true when the demand from the batch is somewhat smaller than the size of the pools, in particular in all realistic cases we are interested in). Hence, arbitrageurs trade on the batch in the same direction as noise traders.

We established that the arbitrageurs collectively purchase y^* ETH with the batch to sell these ETH on the CPAMM. We now complete the characterization of the equilibrium by showing that the first arbitrageur after the batch purchases the full amount.

Suppose not: there is an equilibrium in which n arbitrageurs collectively purchase $y^* = \sum_{i=1}^n y_i^*$ ETH, and the first arbitrageur after the batch purchases $y_1^* \neq y^*$ ETH. In this case, the first arbitrageur's profits are

$$\frac{y_1^* \cdot Q^\$}{Q^E - \frac{2}{3}(A+y^*)} - \frac{y_1^* \cdot Q^\$ Q^E}{(Q^E - \frac{2}{3}(A+y^*))(Q^E - \frac{2}{3}(A+y^*)+y_1^*)}$$

but because $y_1^* \neq y^*$, then

$$\frac{Q^\$}{Q^E - \frac{2}{3}(A+y)} \neq \frac{Q^\$ Q^E}{(Q^E - \frac{2}{3}(A+y)+y)^2}$$

which implies that $y_1^* \neq y^*$ does not maximize the first arbitrageur's profits, who therefore wants to deviate. ◀

Proof of Proposition 7. Call y_1 the amount purchased on the batch by the first arbitrageur, y_2 the amount purchased on the batch by all other arbitrageurs who trade before the batch, y_3 the amount purchased on the batch by the arbitrageur who trades immediately after the batch.

As a first step, we derive how the batch will trade $A + y_1 + y_2 + y_3$. Suppose the batch purchases x_1 on the traditional CPAMM and x_2 on the FM-AMM, with $A + y_1 + y_2 + y_3 = x_1 + x_2$. Total expenditure is

$$\frac{Q^\$ Q^E x_1}{(Q^E + y_1 + y_2)(Q^E + y_1 + y_2 - x_1)} + \frac{Q^\$ x_2}{Q^E - 2x_2}.$$

Which is minimized by trading

$$x_1 = \frac{2}{3}(A + y_3) + y_1 + y_2,$$

on the CPAMM and

$$x_2 = \frac{1}{3}(A + y_3),$$

on the FM-AMM. Intuitively, the batch operator first restores on the CPAMM the liquidity taken by the earlier arbitrageurs and then trades the rest of the batch 1/3 on the FM-AMM and the rest on the CPAMM.

Note that, to have an equilibrium, the sum of all trades must re-align the marginal price on the CPAMM with the price on the FM-AMM, which implies

$$\frac{Q^\$}{Q^E - \frac{2}{3}(A + y_3)} = \frac{Q^\$ Q^E}{(Q^E - \frac{1}{3}(A - y_3))^2}.$$

The problem of the first arbitrageur after the batch is identical to that of the earlier case, because the batch has brought back the CPAMM to its initial state. He will perform the trade that re-aligns the FM-AMM price with that of the CPAMM, and is $y_3 = y^*$ as in Equation (3).

Consider then the problem from the point of view of the first arbitrageur. Because of the behavior of the first arbitrageur after the batch, the first arbitrageur in the block anticipates that the price on the FM-AMM does not depend on his trade. He therefore maximizes profits by buying tokens on the CPAMM until its marginal price is equal to the price on the FM-AMM (and on the batch), which is given by

$$\frac{Q^\$}{Q^E - \frac{2}{3}(A + y^*)} = \frac{Q^\$ Q^E}{(Q^E + y_1^*)^2}$$

or

$$y_1^* = \sqrt{(Q^E)^2 - Q^E \frac{2}{3}(A + y^*)} - Q^E$$

which is negative if $A > 0$ (i.e., the arbitrageur buys on the CPAMM and sells on the batch) and positive if $A < 0$ (i.e., the arbitrageur sells on the CPAMM and buys on the batch). ◀

Strategic Liquidity Provision in Uniswap V3

Zohu Fan
Harvard University, Cambridge, MA, USA

Francisco Marmolejo-Cossio
Harvard University, Cambridge, MA, USA
IOG, USA

Daniel Moroz
Harvard University, Cambridge, MA, USA

Michael Neuder
Harvard University, Cambridge, MA, USA

Rithvik Rao
Harvard University, Cambridge, MA, USA

David C. Parkes
Harvard University, Cambridge, MA, USA

Abstract

Uniswap v3 is the largest decentralized exchange for digital currencies. A novelty of its design is that it allows a liquidity provider (LP) to allocate liquidity to one or more closed intervals of the price of an asset instead of the full range of possible prices. An LP earns fee rewards proportional to the amount of its liquidity allocation when prices move in this interval. This induces the problem of *strategic liquidity provision*: smaller intervals result in higher concentration of liquidity and correspondingly larger fees when the price remains in the interval, but with higher risk as prices may exit the interval leaving the LP with no fee rewards. Although reallocating liquidity to new intervals can mitigate this loss, it comes at a cost, as LPs must expend gas fees to do so. We formalize the dynamic liquidity provision problem and focus on a general class of strategies for which we provide a neural network-based optimization framework for maximizing LP earnings. We model a single LP that faces an exogenous sequence of price changes that arise from arbitrage and non-arbitrage trades in the decentralized exchange. We present experimental results informed by historical price data that demonstrate large improvements in LP earnings over existing allocation strategy baselines. Moreover we provide insight into qualitative differences in optimal LP behaviour in different economic environments.

2012 ACM Subject Classification Computing methodologies → Modeling methodologies; Computing methodologies → Neural networks

Keywords and phrases blockchain, decentralized finance, Uniswap v3, liquidity provision, stochastic gradient descent

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.25

Related Version *Full Version*: `https://arxiv.org/abs/2106.12033`

Funding This work is supported in part by two generous gifts to the Center for Research on Computation and Society at Harvard University, both to support research on applied cryptography and society.

1 Introduction

Decentralized finance (DeFi) is a large and rapidly growing collection of projects in the cryptocurrency and blockchain ecosystem. From May 2019 to May 2023, the TVL (*total value locked*, meaning the sum of all liquidity provided to the protocol) into DeFi protocols

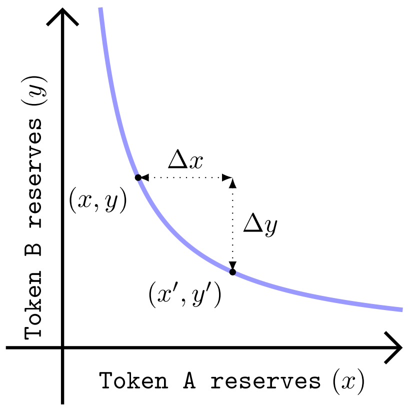


Figure 1 The reserve curve for Uniswap v2. If the pool has reserves (x, y) where x and y represent units of token A and B respectively, then the contract price of token A is $P = y/x$. A trader can send Δx units of token A to receive Δy units of token B, such that $x'y' = L^2$, where $x' = x + \Delta x$ and $y' = y - \Delta y$. The contract price of token A after the trade is $P' = y'/x'$.

has increased 100x from 500 million USD to 50 billion USD. During this time period, TVL rapidly increased in 2021, reaching a peak of roughly 176 billion USD in November 2021, but it also suffered large drops in 2022 leading to current TVL levels.[1]

DeFi aims to provide the function of financial intermediaries and instruments through smart contracts executed on blockchains (typically Ethereum). The importance of decentralized exchanges (DEXes) is that traders can swap tokens of different types without a trusted intermediary. Most DEXes, including Uniswap, fall into the category of *constant function market makers* (CFMMs). Instead of using an order book as is done in traditional exchanges, CFMMs are smart contracts that use an *automated market maker* (AMM) to determine the price of a trade.

In *Uniswap v2*, token pairs can be swapped for each other using *liquidity pools*, which contain quantities of each of the pair of tokens (say, token A and token B). Permitted trades are determined by the *reserve curve*, $xy = L^2$, where x and y denote the the number of tokens of type A and B respectively in the liquidity pool, and the value of L must be maintained across a trade. *Liquidity providers (LPs)* add tokens to liquidity pools for the traders to swap against, and are rewarded through the fees traders pay. An illustrative reserve curve is shown in Figure 1. Assuming that the v2 contract holds x units of token A and y units of token B with $xy = L^2$, then in order to sell some quantity $\Delta x > 0$ of token A for some quantity $\Delta y > 0$ of token B, the trader must keep the product of reserves constant, with Δy such that $(x - \Delta x)(y + \Delta y) = L^2$. This defines an effective contract price for token A in units of token B, i.e., $P = -dy/dx$. In the context of the $xy = L^2$ curve, we have $y = L^2/x \implies P = L^2/x^2 = y/x$. By convention, we take the contract price to be the price of token A, which we may assume is volatile relative to token B. In Uniswap v2, traders pay LPs a fee of 0.3% of the transaction amount in return for using the liquidity to execute a swap [2]. In v2, the liquidity of every LP can be used for swaps at every possible price $P \in (0, \infty)$, and an LP earns fees based on their fraction of the total liquidity in the pool.

Uniswap v3 launched on Ethereum on May 3, 2021 and introduced *concentrated liquidity* [3], where an LP can provide liquidity to each of any number of price intervals, these called *positions*. In particular, liquidity allocated by an LP to position $[P_a, P_b]$ earns fees when the

[1] `https://defillama.com/`

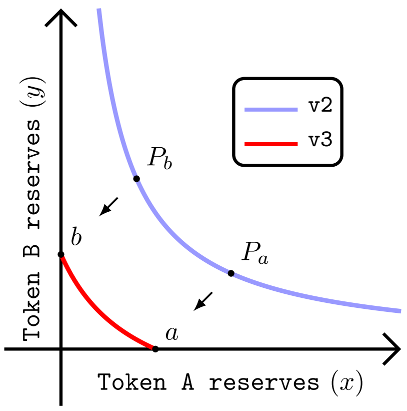


Figure 2 The reserve curve of Uniswap v2 over all prices, and of Uniswap v3 over the price interval $[P_a, P_b]$. When trades give rise to contract prices in this interval, LP assets are swapped according to the v3 curve, which is an affine transformation of the v2 curve and defined to respect the price limits of interval $[P_a, P_b]$.

contract price is in that interval. If multiple LPs allocate liquidity over an interval containing the current price, each is rewarded proportionally to the fraction of the liquidity they own at that price. Figure 2 visualizes the functional invariant respected by the overall assets provided by LPs to support trades over $[P_a, P_b]$. For trades in this interval, a v3 contract effectively shifts the reserve curve of Uniswap v2 via an affine transformation to intercept the axes at a and b, which depend on the end points of interval $[P_a, P_b]$. This shifted curve is governed by the equation: $\left(x + L/\sqrt{P_b}\right)\left(y + L\sqrt{P_a}\right) = L^2$, and the intercepts, a and b, can be calculated by letting x or y equal zero respectively [3] . This description of Uniswap v3 is inherently local, as it describes trade dynamics for a specific price interval. Gluing the local dynamics together for all prices gives rise to an *aggregate reserve curve* that governs arbitrary trades across all possible prices. This global reserve curve is in turn a function of the aggregate distribution of liquidity provided by all LPs. From the perspective of traders, when more liquidity is allocated to a given price interval, there is less trade slippage for prices in that interval. This reduced slippage corresponds to a flatter section of the aggregate reserve curve, as visualized in Figure 3.

Uniswap v3 supports a diversity of LP strategies in regard to the allocation of liquidity, as an LP can mint multiple positions, each on a different interval. Each LP is presented with a tradeoff between choosing large positions that cover many possible prices but earn less fees and smaller more concentrated positions that are more risky (since they cover fewer prices). Additionally, an LP can reallocate its liquidity as prices change, but this comes at two costs. First, an LP may need to trade between assets to mint new liquidity positions in a reallocation, potentially suffering losses from slippage in such trades. Second, since liquidity allocations are transactions that must be written as updates to the contract and included in a block, they also incur gas fees. Both of these costs must be incorporated in more complex LP strategies that make use of liquidity reallocation. Given the increased complexity in LP actions from v2 to v3, it is important to understand potential ways LPs can benefit from strategically allocating liquidity as prices change, as this ultimately impacts the performance of v3 contracts as DEXs.

In this regard, much relevant work has studied ways in which LPs in Uniswap v3 can optimize their earnings when faced with uncertain beliefs over how trades will evolve over time. Most relevant to our work are two papers, [11, 14], the first of which provides insight

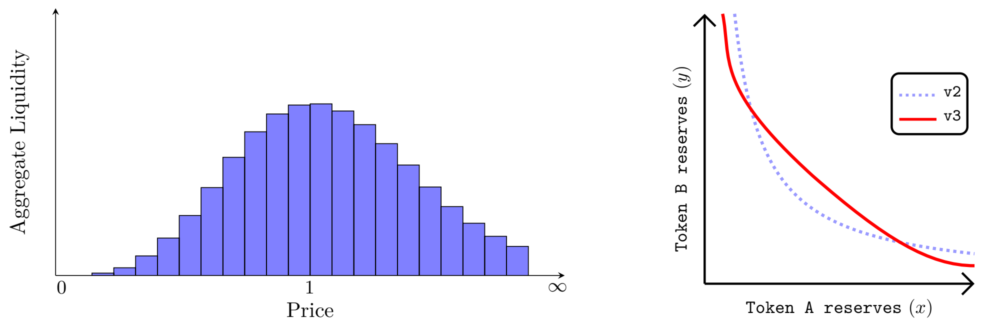


Figure 3 An aggregate distribution of liquidity for a Uniswap v3 contract (left plot), with most liquidity allocated close to unit price $P = 1$. This results in an aggregate reserve curve (right, red line), which is flatter than the corresponding v2 curve (dotted blue) at prices close to $P = 1$ and supports a larger volume of trades at these prices with less slippage.

into how LPs can profit from liquidity reallocations with simple positions, and the second of which focuses on how LPs can profit from complicated static liquidity positions over a given time horizon.[2] In more detail, the authors of [11] provide a closed form solution for computing optimal LP allocations that dynamically readjust positions to different intervals as prices evolve over time. Though these strategies make use of dynamic reallocation of liquidity, the only allocations explored in the optimization are individual v3 positions over a single price range (which forcibly change at each reallocation) instead of the full class of potential allocations available to an LP in Uniswap v3. The authors of [14] compute optimal arbitrary v3 positions for small LPs who seek to maximize profit and loss over a fixed time horizon, but only consider static LP strategies which do not make use of reallocations as prices change.

Our work addresses the gaps from these papers by exploring optimal liquidity provision strategies that simultaneously make use of liquidity reallocations and the full complexity of v3 positions. As in [14], we adopt the perspective of an LP with stochastic beliefs over how market prices evolve over a given time horizon, and how contract prices may change along with market prices via arbitrage and non-arbitrage trades. In addition, we also make the assumption that the LP is small enough that these beliefs are independent of capital allocated by the LP. We provide an optimization framework for computing optimal dynamic liquidity provision strategies over a given time horizon, and we show that such strategies can provide large gains to LPs over strategies that are static or strategies that make use of simple liquidity positions.

We define *dynamic liquidity provision strategies* as a means by which LPs can mitigate potential losses by using earned capital to reallocate liquidity to different price intervals in a given time horizon. In particular, we focus on the family of *τ-reset strategies*, which allocate over an interval of prices centered on the current price and whose width is controlled by τ (including the possibility of declining to allocate liquidity) and reallocate whenever the price moves outside this interval. Moreover, within the space of potential dynamic liquidity provision strategies, we distinguish between those that are context-aware and context-independent depending on whether they incorporate historical price and contract information in their decision-making. We develop methods of stochastic optimization for optimizing over τ-reset strategies when an LP has stochastic beliefs on market and contract

[2] Both of these papers were written after the first version of the present paper [21]

prices, and with different levels of risk-aversion. We give empirical results based on historical Ethereum price data to show that incorporating both of these aspects into LP allocation strategies gives rise to large gains in performance for LPs of various levels of risk-aversion, especially for context-aware reallocation strategies, which we optimize for with a neural network. In addition, our results provide insight into how optimal LP behaviour varies depending on relevant aspects of their economic environment. In particular, we find that more risk-averse LPs spread their liquidity over larger price ranges, especially when faced with a larger volume of non-arbitrage trades. In addition we also find that, as expected, optimal reset frequencies are sensitive to the reallocation costs.

1.1 Related work

The Uniswap v1 protocol was defined by [1], followed up with v2 by [2], and most recently amended in v3 by [3]. There has been a growing body of work studying LP incentives in Uniswap v2. [6] present an analysis of Uniswap, and more broadly of constant-product AMMs, and demonstrate conditions for which the markets closely track the price on an external reference market. [4] extend this line of research by demonstrating that the more general class of CFMMs incentivize participants to report the true price of an asset on an external market, demonstrating their value as price oracles.

[7] study the equilibrium liquidity provision of constant product AMMs, and show that strategic LPs in the Uniswap v2 environment may have a non-monotonic best response when parameterized with the opponents' liquidity provision. [5] extend this line of work to CFMMs and calculate bounds on the LP rewards based on the curve that defines the CFMM. [10] studies the adopation of decentralized exchanges more generally, using a sequential game framework to model interactions between LPs and traders. In addition, [23] and [15] provide an axiomatic framework for general CFMMs similar in nature to Uniswap v2, with the latter focusing on connections with CFMMs in the prediction market setting. [12] consider *geometric mean market makers* (G3Ms), and show that passive liquidity provision can be used to replicate payoffs of financial derivatives and more active trading strategies. [24] and [25] analyze the growth in wealth of an LP in CFMM for a geometric Brownian motion price process. [13] extend this to more general LP objectives and diffusion processes.

All above work applies to Uniswap v2 and similarly structured CFMMs but not to v3. An early blog post by [18] describes a "passive rebalancing" strategy for v3, which aims at maintaining a 50-50 ratio of value for the assets of the LP. In addition to [14] and [11], further related work on v3 includes [20], who decompose divergence/impermanent loss into hedgeable market risk and profit made by arbitrage traders at a loss to the exchange. (This is related to [11], who decompose divergence/impermanent loss into two components: the loss due to arbitrage (convexity cost) and the cost of locking capital). [9] uses regret-minimization from online learning to provide liquidity provision strategies under adversarial trading. [19] and [16] study the construction of optimal CFMMs from the perspective of LP beliefs, with the latter providing a Myersonian framework for creating incentive compatible AMMs, and the former employing techniques from convex optimization to determine optimal trading functions based on LP beliefs on future trades. [17] study the risks inherent to LP returns for multiple fixed strategies in different economic environments, concluding that liquidity provision in v3 is a sophisticated game where uninformed retail traders can suffer large disadvantages relative to more informed agents. In addition to studying optimal static liquidity provision, [14] provide insights on how aspects of a v3 contract, notably the partition of price space, have implications on LP profit as well as gas fees incurred by traders.

1.2 Outline

Section 2 introduces the Uniswap v3 protocol. Section 3 formalizes the earnings to an LP from a dynamic liquidity provision strategy and introduces the family of τ-reset strategies. Section 4 introduces the computational methods for optimizing over τ-reset strategies and specifically defines the context-aware/independent dynamic liquidity strategies we empirically study as well as simple baselines to which we compare performance. Section 5 provides details regarding the economic environments we modulate to study optimal LP behaviour, and Section 6 presents empirical results. Section 7 gives open problems for future research and concludes.

2 The Mechanics of Uniswap

In this section, we provide a brief overview of Uniswap v3 contracts. In all that follows, we consider a v3 contract that enables trades between two types of tokens that we designate token A and token B. Furthermore, without loss of generality, we assume token B is the numeraire, hence when we refer to the price in the contract, we refer to the price of a unit of A in terms of B. As mentioned in the introduction, liquidity providers (LPs) provide bundles of A and B tokens to the contract as liquidity to be traded against. The following sections largely follow the mathematical formalism of [14].

2.1 v3 Contracts

Partitioned Price Space

A Uniswap contract maintains the *contract price* of token A given by $P \in (0, \infty)$. This is the infinitesimal price that traders obtain for trading with the contract. In Uniswap v2 contracts, the liquidity that LPs provide is used to support every trade, whatever the trade price. Uniswap v3 contracts provide a richer set of actions for an LP, where they can specify a price range where liquidity is to be used for trading. In order to enable this functionality, a v3 contract partitions token A prices into a finite set of *price buckets*: $\boldsymbol{\mu} = \{B_{-m}, \ldots, B_0, \ldots, B_n\}$ with buckets $B_i = [a_i, b_i]$. We also require $a_0 < 1 < b_0$, so that the parity price lies in the 0-th bucket, and that $b_i = a_{i+1}$ for $i \in \{-m, \ldots, n-1\}$.

Minting and Burning Liquidity

LPs provide (or *mint*) liquidity in a particular bucket, $B_i = [a_i, b_i]$, referred to as *B_i-liquidity*, by sending a bundle of A and B tokens to the contract. The token bundle required to mint L units of B_i-liquidity is given by the *liquidity value function* [14], and is tuple $\mathcal{V}^{(3)}(L, P, B_i)$, where the first component is the quantity in token A and the second is the quantity in token B. For $a < b$, let $\Delta^x_{b,a} = \frac{1}{\sqrt{a}} - \frac{1}{\sqrt{b}}$ and $\Delta^y_{a,b} = \sqrt{b} - \sqrt{a}$.

▶ **Definition 1** (B_i-Liquidity Value [14])**.** *For contract price P, bucket $B_i = [a_i, b_i]$, and number of units $L > 0$, the* bundle liquidity value *$\mathcal{V}^{(3)}(L, P, B_i) \in \mathbb{R}^+ \times \mathbb{R}^+$ is defined as*

$$\mathcal{V}^{(3)}(L, P, B_i) = \begin{cases} (L\Delta^x_{b_i,a_i}, 0) & \text{if } P < a_i, \\ (0, L\Delta^y_{a_i,b_i}) & \text{if } P > b_i, \text{ or} \\ (L\Delta^x_{b_i,P}, L\Delta^y_{a_i,P}) & \text{if } P \in B_i, \end{cases} \tag{1}$$

and specifies the bundle of A and B tokens, respectively, to mint L units of B_i-liquidity.

LPs can also remove liquidity they have a claim to from the contract by means of the value function. When this happens, we say that an LP *burns* L units of B_i liquidity, and the LP receives token bundle $\mathcal{V}^{(3)}(L, P, B_i)$ if the contract price is P. Since $\mathcal{V}^{(3)}$ is linear in L, we also adopt shorthand $\mathcal{V}^{(3)}(P, B_i) = \mathcal{V}^{(3)}(1, P, B_i)$ for the token bundle value of a single unit of liquidity, with $\mathcal{V}^{(3)}(L, P, B_i) = L \cdot \mathcal{V}^{(3)}(P, B_i)$.

Trading and Fees

When the contract price is in a given bucket, the trade dynamics respect the shifted reserve curve of Figure 2. Fees are governed by a fee rate, $\gamma \in (0, 1)$, such that if a given trader sends Δx units of token A to the contract, $\gamma \Delta x$ is first skimmed as fees to be shared proportionally amongst LPs who have allocated liquidity to the bucket containing the current price. The remaining $(1 - \gamma)\Delta x$ units of token A are used for trading via the v3 reserve curve. We refer the reader to the full version of the paper for a detailed discussion on trading and fees in Uniswap v3.

3 Liquidity Allocation Strategies and LP Earnings

In this section, we describe a rich set of strategies that LPs can use to maximize their earnings over a given time horizon. As token B is the numeraire, we measure all earnings in terms of units of token B and we assume that the LP begins with a fixed budget consisting of $W > 0$ units of token B. We model price discovery between A and B tokens as occurring outside of Uniswap contracts, and in addition to the contract price there is a *market price* that is determined by external markets. We denote the market price by P_m and contract price by P_c, and we extend price P so that $P = (P_m, P_c)$ denotes a *contract-market price pair*. Furthermore, we assume that arbitrage trade can be performed by traders at price P_m which in turn brings P_c close to P_m (see Section 5.1). In what follows, we consider time horizons that are characterized by a single sequence of $T > 0$ contract-market prices, denoted by $\mathbf{P} = (P_0, \ldots, P_T)$, where $P_t = (P_{c,t}, P_{m,t})$ is the t-th contract-market price in the sequence (time steps are indexed t).

3.1 Static Liquidity Provision Strategies

We begin by using a similar mathematical formalism and notation from [14] to express an LP's earnings over price sequence $\mathbf{P}$ for a simple family of liquidity allocation strategies.

▶ **Definition 2** (Static liquidity provision strategy). *An LP with an initial budget of $W > 0$ units of token B uses a* static liquidity provision strategy *when they*

1. *mint an initial liquidity allocation at P_0,*
2. *accrue token fees over the course of $\mathbf{P}$ as prices change, and*
3. *burn the existing liquidity allocation at P_T, the end of the time horizon, to recover invested capital from the contract.*

We focus on a single LP, and suppose they mint liquidity positions at the beginning of $\mathbf{P}$, when contract-market prices are given by $P_0 = (P_{c,0}, P_{m,0})$, with their initial budget of $W > 0$ units of token B. For this, let $\mathbf{x} = (x_{-m}, \ldots, x_n)$ denote a *proportional liquidity allocation*, where for $i \in \{-m, \ldots, n\}$, $x_i \geq 0$ represents the proportion of capital used to mint B_i-liquidity (with $\sum_{i=-m}^{n} x_i \leq 1$ so that $\mathbf{x} \in \Delta^{m+n+2}$, the $(m + n + 2)$-dimensional simplex). The LP uses $W x_i$ units of token B to mint B_i-liquidity for each of $i \in \{-m, \ldots, n\}$. Let $x_{n+1} = 1 - \sum_{i=-m}^{n} x_i \in [0, 1]$ denote the proportion of capital that the LP does not invest and keeps as units of token B.

Let $\mathcal{B} : (\mathbb{R}^+)^2 \times \mathbb{R}^+ \to \mathbb{R}^+$, defined as $\mathcal{B}((z_1, z_2), P_m) = P_m \cdot z_1 + z_2$, return the *token B market worth* of a bundle of A and B tokens when token A has market price P_m. For a given contract-market price sequence, $\mathbf{P}$, let $w_i = \mathcal{B}(\mathcal{V}^{(3)}(P_{c,0}, B_i), P_{m,0})$ denote the amount of B tokens required to mint one unit of B_i-liquidity at the initial contract-market price of $P_0 = (P_{c,0}, P_{m,0})$. With this in hand, let vector $\boldsymbol{\ell} = (\ell_{-m}, \ldots, \ell_n)$ denote the *absolute liquidity allocation* induced by initial budget (W), proportional liquidity allocation $(\mathbf{x})$ and initial contract-market price (P_0). It follows that $\ell_i = \frac{Wx_i}{w_i}$ units of B_i-liquidity for each $i \in \{-m, \ldots, n\}$. This implies that $\boldsymbol{\ell}$ is linear as a function of each of $\mathbf{x}$ and W.

3.1.1 Linearity of Fee Rewards in $\mathbf{x}$

We are ultimately interested in expressing an LP's token B value of earnings as a function of their liquidity allocation over the contract-market price sequence. We begin by determining the amount of trading fees earned by an LP. Interestingly, these earnings are not only independent of other LP allocations, but also linear in $\boldsymbol{\ell}$ (and consequently $\mathbf{x}$).

▶ **Theorem 3** (Section 3.1 [26]). *For a fixed contract-market price sequence* $\mathbf{P}$*, the amount of A tokens and B tokens accrued from fees is linear in* $\boldsymbol{\ell}$ *and independent of the liquidity of other LPs in the contract.*

That the fees that a single LP earns are independent of other LPs' liquidity allocations follows from the assumption that contract-market prices are independent of the liquidity allocation of this LP. Indeed, for a fixed price sequence, allocating liquidity by an LP has two effects. First, increasing the liquidity means that a larger volume of trade needs to happen to effect the same price change, resulting in more fees to be paid out to LPs. Second, the same LP has a proportionally larger amount of liquidity across the relevant price interval. The net effect is that fees are only a function of a single LP's proportional (or absolute) liquidity allocation. Theorem 3 justifies the following definition of a trading fee function for a single LP and a given contract-market price sequence.

▶ **Definition 4** (Trading Fee Functions). *Suppose that* $\mathbf{P}$ *is a fixed contract-market price sequence and* $W > 0$ *an initial token B budget. For a proportional (or absolute) liquidity allocation given by* $\mathbf{x}$ *(or* $\boldsymbol{\ell}$*), we let* $F^A(\mathbf{x}, W, \mathbf{P})$ *(or* $F^A(\boldsymbol{\ell}, \mathbf{P})$*) denote the units of A tokens earned from fees over* $\mathbf{P}$ *from downward price movements. Similarly, we let* $F^B(\mathbf{x}, W, \mathbf{P})$ *(or* $F^B(\boldsymbol{\ell}, \mathbf{P})$*) denote the units of B tokens earned from fees over* $\mathbf{P}$ *from upward price movements.*

When the resulting absolute liquidity allocation $\boldsymbol{\ell}$ is treated as a function of $\mathbf{x}$ and W, it is linear in $\mathbf{x}$ and W, and it follows that both $F^A(\mathbf{x}, W, \mathbf{P})$ and $F^B(\mathbf{x}, W, \mathbf{P})$ are linear in $\mathbf{x}$ and W. For this reason, we let $F^A(\mathbf{x}, \mathbf{P}) = F^A(\mathbf{x}, 1, \mathbf{P})$ and $F^B(\mathbf{x}, \mathbf{P}) = F^B(\mathbf{x}, 1, \mathbf{P})$. This in turn implies that $F^A(\mathbf{x}, W, \mathbf{P}) = W \cdot F^A(\mathbf{x}, \mathbf{P})$, and similarly $F^B(\mathbf{x}, W, \mathbf{P}) = W \cdot F^B(\mathbf{x}, \mathbf{P})$ for arbitrary $\mathbf{x}, W$, and $\mathbf{P}$.

3.1.2 Burning Liquidity Allocations at P_T

All that remains to fully quantify the earnings of an LP over $\mathbf{P}$ is to take into account the capital they obtain by burning their liquidity positions at time T under contract-market price $P_T = (P_{c,T}, P_{T,m})$, obtaining a final quantity of token B. For this, let $w_i' = \mathcal{B}(\mathcal{V}^{(3)}(P_{c,T}, B_i), P_{m,T})$ be the token B worth of the capital obtained from burning 1 unit of B_i-liquidity at the final contract-market price of $P_T = (P_{c,T}, P_{m,T})$. Given absolute liquidity position $\boldsymbol{\ell}$, the overall token B value of capital obtained from burning is $C(\boldsymbol{\ell}, \mathbf{P}) =$

$\sum_{i=-m}^{n} w_i' \ell_i$, and linear in $\boldsymbol{\ell}$. Let $C(\mathbf{x}, W, \mathbf{P})$ denote the final token B worth (at P_T) of a liquidity position minted at P_0 with $\mathbf{x}$ and budget W. Since proportional allocations also allow an LP to maintain funds in terms of token B (i.e., when $x_{n+1} > 0$), we obtain the expression $C(\mathbf{x}, W, \mathbf{P}) = C(\boldsymbol{\ell}, \mathbf{P}) + W x_{n+1}$, where $\boldsymbol{\ell}$ is the absolute liquidity allocation corresponding to $\mathbf{x}$. Once more, since $\boldsymbol{\ell}$ is in turn linear in $\mathbf{x}$ and W, it follows that C is linear in $\boldsymbol{\ell}$ and W. For this reason, we let $C(\mathbf{x}, \mathbf{P}) = C(\mathbf{x}, 1, \mathbf{P})$, so that $C(\mathbf{x}, W, \mathbf{P}) = W \cdot C(\mathbf{x}, \mathbf{P})$ for arbitrary $\mathbf{x}, W$, and $\mathbf{P}$.

3.1.3 Linearity of Overall Earnings in $\mathbf{x}$

We now define an LP's earnings over a contract-market price sequence.

▶ **Definition 5.** *Suppose that $\mathbf{P} = (P_0, \ldots, P_T)$ is a contract-market price sequence and that $\mathbf{x} \in \Delta^{m+n+2}$ is a proportional liquidity allocation. An LP's earnings (in units of token B) under $\mathbf{P}$ with an initial budget of $W > 0$ is,*

$$V(\mathbf{x}, W, \mathbf{P}) = P_{m,T} \cdot F^A(\mathbf{x}, W, \mathbf{P}) + F^B(\mathbf{x}, W, \mathbf{P}) + C(\mathbf{x}, W, \mathbf{P}).$$

From this definition, we conclude that an LP's earnings from a fixed contract-market price sequence and with a static liquidity provision strategy are linear in $\mathbf{x}$ and W.[3]

▶ **Theorem 6.** *V is linear in both W and $\mathbf{x}$ for any contract-market price sequence, $\mathbf{P}$.*

Proof. This is an immediate corollary of the fact that F^A, F^B and C are each linear in $\mathbf{x}$ and W for any contract-market price sequence $\mathbf{P}$. ◀

Similar to before, we use the shorthand $V(\mathbf{x}, \mathbf{P}) = V(\mathbf{x}, 1, \mathbf{P})$ such that $V(\mathbf{x}, W, \mathbf{P}) = W \cdot V(\mathbf{x}, \mathbf{P})$ for arbitrary $\mathbf{x}, W$, and $\mathbf{P}$.

3.2 Dynamic Liquidity Provision Strategies

In this section, we introduce the notion of *dynamic liquidity provision* strategies, where an LP can reallocate their liquidity at any time step of the contract-market price sequence, $\mathbf{P}$.

At a high level, if an LP chooses to reallocate liquidity at time t, they burn their existing liquidity position and use their overall earnings at time t, denoted by W_t, to mint a new proportional allocation, $\mathbf{x}$, given the contract-market price P_t. Reallocation comes at a cost however, which represents the fact that burning and minting positions on a Uniswap contract requires paying gas fees, and that minting new positions may require the LP to trade between A and B tokens. We model reallocation costs as proportional to overall earnings used to mint the position $\mathbf{x}$ (i.e. the funds corresponding to capital kept token B outside the contract (x_{n+1}) do not incur a cost). Cost is specified by a single parameter, $\eta \in [0, 1]$, such that the LP retains $\eta W_t(1 - x_{n+1})$ of the funds they intend to use for minting a new position after paying reallocation costs at time t (i.e., by paying $(1 - \eta)W_t(1 - x_{n+1})$ in reallocation cost).

We partition price sequence $\mathbf{P}$ into *epochs*, which are sequences of contract-market prices from $\mathbf{P}$ uninterrupted by liquidity reallocations. For an LP that burns and reallocates liquidity positions at time steps $\mathbf{t} = (t^0, \ldots t^k)$, where $t^0 = 0$ and $t^k = T$, there are $k \geq 1$ epochs, where the j-th epoch is $E^j = (P_{t^j}, \ldots, P_{t^{j+1}})$. When indexing over epochs we use superscripts, and when indexing over time-steps in $\mathbf{P}$ we use subscripts.

[3] As a side note, with the definition of LP's earnings here we are modeling the profit and loss (PnL) of an LP who holds their Uniswap v3 liquidity position without hedging. The strategy optimization approach in this paper can be naturally extended for maximizing the PnL of a delta-hedged LP as well by incorporating Loss versus Rebalancing (LVR) from [20] since LVR is also linear in $\mathbf{x}$ and W.

Each epoch, E^j, is associated with the total earnings, W^j, the LP has accrued at the beginning of the epoch, and the proportional liquidity allocation, $\mathbf{x}^j$, the LP uses to mint positions with W^j over E^j. From the static liquidity allocation analysis, it follows that the LP's earnings over the epoch are given by $W^j \cdot V(\mathbf{x}^j, E^j)$. After incorporating the proportional reallocation cost, the earnings available for the LP to use for E^{j+1} are $W^{j+1} = \eta W^j \cdot V(\mathbf{x}^j, E^j)(1 - x_{n+1}^{j+1})$. We encode the collection of all proportional allocations as a $(k \times (m + n + 1))$ matrix $\mathbf{X}$, such that the j-th row of $\mathbf{X}$ corresponds to $\mathbf{x}^j$.

▶ **Definition 7** (Dynamic liquidity provision strategy). *We say that* Λ *is a* dynamic liquidity provision strategy *if it takes as input a contract-market price sequence,* $\mathbf{P} = (P_1, \ldots, P_T)$*, and defines:*

- $\mathbf{t} = (t^0, \ldots, t^k)$*, with* $k \geq 1$*,* $t^0 = 0$*, and* $t^k = T$*. These are time-steps where a reallocation occurs.*
- $\mathbf{X} \in mat\,(k \times (m + n + 1))$ *such that the* j*-th row of* $\mathbf{X}$ *encodes* $\mathbf{x}^j$*, the proportional liquidity allocation profile to be used at* E^j*.*

We write $\Lambda(\mathbf{P}) = (\mathbf{t}, \mathbf{X})$ *and say that this is the* realized dynamic liquidity provision strategy *of an LP under* Λ *for contract-market price sequence* $\mathbf{P}$*.*

For an initial budget $W(= W_0 = W^0)$, we let $V(\Lambda, W, \mathbf{P})$ denote the overall earnings an LP obtains over $\mathbf{P}$ by employing strategy Λ, which can be computed recursively over the epochs of $\mathbf{P}$. As in previous sections, we let $V(\Lambda, \mathbf{P}) = V(\Lambda, 1, \mathbf{P})$.

3.2.1 Reset Liquidity Strategies

In practice, a strategy Λ may not be implementable, for example requiring an LP to know the full contract-market price sequence, $\mathbf{P}$, before it is realized. In this section, we focus on a specific family of implementable dynamic liquidity provision strategies, the *reset liquidity strategies.* For this, an LP at time-step t with accumulated earnings W_t may choose to *trigger* a liquidity reallocation based on the contract-market price sequence up to time t, denoted $\mathbf{P}_{\leq t}$. For reset liquidity strategies, the LP maintains a *reference bucket index* $Z_t \in \{-m, \ldots, n\}$ (correspondingly a reference bucket $B_{Z_t} \in \boldsymbol{\mu}$). We let $S_t = (Z, W_t, \mathbf{P}_{\leq t})$ denote the *system state*, and we let $\mathcal{S}$ denote the space of all possible system states. A liquidity reset consists in updating the reference bucket index and using the W_t units of B tokens at their disposal to mint a liquidity position relative to the reference bucket index Z_t.

▶ **Definition 8** (Reset liquidity provision strategy). *A* reset liquidity provision strategy (reset-LP strategy) *is composed of:*

1. *A* reference bucket update function*,* g*, which takes as input an arbitrary system state* $S \in \mathcal{S}$ *and updates the reference bucket index to* $Z \leftarrow Z'$ *where* $Z' = g(S)$*.*
2. *An* allocation function*,* $A : \mathcal{S} \times \mathbb{Z} \to [0, 1]$*, which specifies the fraction of budget an LP allocates to mint liquidity in each bucket relative to* Z *after a liquidity reset is triggered. More specifically,* A *gives rise to the proportional allocation* $\mathbf{x}$ *such that* $x_i = A(S, i - Z)$*.*
3. *A* reset condition*,* $h(S) \in \{0, 1\}$*, which is an indicator function for whether a reset is triggered in system state* $S \in \mathcal{S}$ *and specifies which contract-market prices, relative to the reference bucket, will trigger a liquidity reset. In the event of a trigger, a new reference bucket is computed via update function* g*.*

We denote a reset-LP strategy by tuple (g, h, A)*.*

Of particular interest is the family of τ*-reset strategies.* These strategies have LPs reset liquidity when the index of the bucket containing the contract price is more than τ away from the reference index, Z. In the case of a reset, the reference bucket changes to the bucket containing the current contract price.

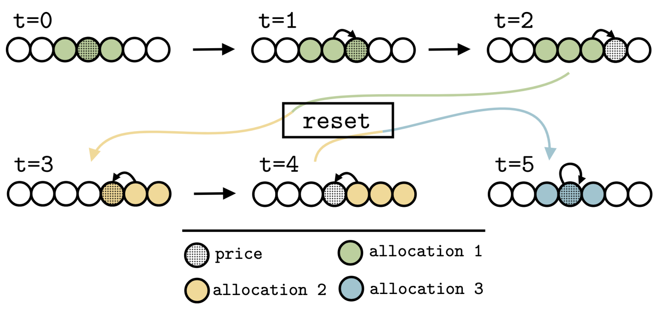


Figure 4 An illustration of how a τ-reset strategy with $\tau = 1$ can play out. Buckets are represented by circles, and for simplicity we assume that market and contract prices move together at each time step. The shaded circle represents the bucket that contract/market prices are in, and the dynamics of price movements are expressed by the smaller arrows between buckets. Colored buckets represent the contiguous $2\tau + 1$ buckets centered around an epoch's reference bucket. For this sequence, we see that price movements at $t_1 = 2$ and $t_2 = 4$ trigger resets, as the shaded bucket escapes the contiguous $2\tau + 1$ colored buckets. The specific reallocation after each trigger is specified by the allocation function A in the τ-reset strategy.

▶ **Definition 9** (τ-reset Strategy). *Suppose that τ is a non-negative integer. We let $h_\tau : \mathcal{S} \to \{0, 1\}$ and $g_\tau : \mathcal{S} \to \{-m, \ldots, n\}$ denote trigger and reference bucket update functions, defined for system state $S_t = (Z_t, W_t, \mathbf{P}_{\leq t}) \in \mathcal{S}$ as*

- *$h_\tau(Z_t, W_t, \mathbf{P}_{\leq t}) = 1$ if and only if $P_{c,t} \in B_i$ and $|Z_t - i| > \tau$, and*
- *$g_\tau(Z_t, W_t, \mathbf{P}_{\leq t}) = P_{c,t}$.*

We say that (g_τ, h_τ, A) is a τ-reset strategy*, for any allocation function, $A : \mathcal{S} \times \mathbb{Z} \to [0, 1]$.*

We illustrate the versatility of τ-reset strategies through some examples:

1. (Static Strategies): For $\tau > T$, i.e., the time horizon of $\mathbf{P}$, we recover static strategies.
2. (Uniform τ-Reset Strategies): Allocating liquidity uniformly on a range of contiguous buckets centered around the current reference bucket B_{Z_t} and resetting when prices move outside of this range.
3. (Context-Independent Allocation Strategies): Setting $A(S, i) = A_i \in \mathbb{R}$ for all $S \in \mathcal{S}$; i.e., the proportional allocations relative to baseline bucket index are always the same at the time of a reset trigger.

4 Optimizing Earnings

In this section, we formulate the earnings optimization problem faced by an LP with *belief, $\mathcal{P}$,* defining a distribution on contract-market price sequences in a given time horizon. In the most general case, belief $\mathcal{P}$ would depend on the liquidity allocation strategy used by an LP. For example, if the LP provides a large amount of liquidity for a given price interval, this would in turn reduce the slippage of trades at those prices, which may in turn increase the volume of trades facilitated by the contract, and hence change $\mathcal{P}$. As in [14], we make the simplifying assumption, reasonable for a small LP, that their belief $\mathcal{P}$ is a *liquidity-independent distribution*, and independent of the strategic liquidity strategy used by the LP. Going forward, we limit our attention to liquidity-independent beliefs. In particular, we will model non-arbitrage traders who trade to a particular buy or sell contract price whose value is unaffected by this LP's liquidity allocation, along with arbitrage traders whose trades are triggered by considerations of market price vs contract price.

4.1 Optimal τ-reset Strategies

We've seen that τ-reset strategies are a versatile framework for dynamic liquidity provision. For a given value of τ, the only choice in defining a τ-reset strategy is the allocation function A, and we let $\Lambda_\tau(A) = (g_\tau, h_\tau, A)$ denote the resulting τ-reset strategy.

In this section, we provide a means of optimizing expected earnings for a given τ. For this, we assume that $A \in \mathcal{A}$, where $\mathcal{A}$ is a family of allocation functions. The space of all allocation functions is large, with an allocation function potentially depending on the entire history of contract-market price sequences and LP actions up to the point when a liquidity reset is triggered.

In defining an LP's optimization problem, we consider LPs with different levels of *risk-aversion*, encoded by a *utility function*, $u : \mathbb{R} \to \mathbb{R}$ (we provide example utility functions below), and we assume that an LP wants to select an allocation function to maximize $V^u_{\tau,\mathcal{P}}(A) = \mathbb{E}_{\mathbf{P}\sim\mathcal{P}}\left[u(V(\Lambda_\tau(A), \mathbf{P})\right]$. With this in hand, we let $OPT(\tau, \mathcal{P}, u, \mathcal{A}) = \max_{A\in\mathcal{A}} V^u_{\tau,\mathcal{P}}(A)$, and denote an allocation function in family $\mathcal{A}$ that achieves optimal earnings by A^*.

In general, convex u and concave u correspond to risk-seeking and risk-averse LPs, respectively, and linear u corresponds to a risk-neutral LPs (where we can adopt $u(x) = x$ without loss of generality). Going forward, we adopt as the utility function that with constant Arrow-Pratt measures of absolute risk-aversion [8, 22].

▶ **Definition 10** (Constant Absolute Risk Aversion Utility). *For a given $a \in \mathbb{R}$, the* constant absolute risk aversion utility function*, $u_a : \mathbb{R} \to \mathbb{R}$, is given by:*

$$u_a(x) = \begin{cases} \left(1 - e^{-ax}\right)/a & \text{if } a \neq 0, \text{ and} \\ x & \text{otherwise.} \end{cases} \tag{2}$$

For $a < 0$, $a = 0$, and $a > 0$, utility function u_a models a risk-averse, risk-neutral, and risk-seeking agent, respectively.

4.2 Sampling to Approximate OPT

In order to optimize $V^u_{\tau,\mathcal{P}}(A)$, we approximate the objective by taking a discrete sample of paths from $\mathcal{P}$. As such, suppose that $\mathbf{P}_1, \ldots, \mathbf{P}_N \sim \mathcal{P}$. We define the empirical average earnings of an LP given the sample paths as $\hat{V}^u_\tau(A \mid \mathbf{P}_1, \ldots, \mathbf{P}_N) = \frac{1}{N}\sum_{q=1}^N u(V(\Lambda_\tau(A), \mathbf{P}_q)$. Going forward, we approximate $OPT(\tau, \mathcal{P}, u, \mathcal{A})$ by taking sufficiently many samples from $\mathcal{P}$ and optimizing $\hat{V}^u_\tau$.

4.3 Computing Optimal τ-reset Strategies with Neural Networks

We compute optimal τ-reset strategies by letting the allocation function, A, be parametrized by a feedforward neural network (NN) with parameters given by $\theta \in \boldsymbol{\theta}$. We let A_θ denote the specific allocation function for a given parameter choice $\theta \in \boldsymbol{\theta}$ and we let $\mathcal{A}_{\boldsymbol{\theta}}$ denote the space of all possible parametric settings of the NN. Our objective is to maximize $\hat{V}^u_\tau(A_\theta \mid \mathbf{P}_1, \ldots, \mathbf{P}_N)$ for a given sample of contract/market price paths $\mathbf{P}_1, \ldots, \mathbf{P}_N \sim \mathcal{P}$.

When a reallocation is triggered at the beginning of epoch $j (j = 1, 2, \ldots, k)$, the NN takes as input a set of features C^j that contains context information. The set of context features includes the current time step, the current wealth, the current pool price, the current bucket that the pool price lies in, and an exponentially-weighted moving average (the smoothing parameter value is 0.1) of non-arbitrage trade volume that a hypothetical 1 unit of liquidity over the entire price range would have achieved given the price trajectory.

We use a fully connected neural network architecture with 5 hidden layers, and the size of each hidden layer is $m = 16$. The size of the input layer is $n = 5$ (the number of context features), and the output layer is of size $s = 2\tau + 2$ (the first $2\tau + 1$ dimensions are the proportional capital to be allocated into the corresponding buckets, and there is also a special dimension for not allocating some of the wealth if needed). All the hidden layers are associated with the ReLU activation function. In addition, a soft-max function is added for the final output in order to produce a vector of sum 1. The architecture we use is visualized in the right image of Figure 5.

Unpacking the objective, $\hat{V}^u_\tau(A_\theta \mid \mathbf{P}_1, \ldots, \mathbf{P}_N)$ shows a fundamental recurrence in the given allocation function A_θ. This is because an allocation produced by A_θ is deployed into the pool and affects the value of wealth when the next reallocation is triggered, and wealth is used as part of the input to A_θ for the new reallocation as visualized in the left diagram of Figure 5. However, the NN representation of A allows gradients to be pushed through the recurrence with standard back propagation methods used for recurrent neural networks. Given this, we find optimal $\theta \in \boldsymbol{\theta}$ via standard gradient descent methods.

In more detail, for optimization of the NN (ODRA) and the constant allocation (OIRA)[4], we use stochastic gradient descent based on sampled price trajectories. The number of training steps is 10000 for both optimize methods. The learning rate for the NN is 10^{-3} while the learning rate for the constant allocation is 10^{-2}. In addition, the Adam optimizer is used for both methods.[5]

As risk aversion parameter a increases, the relative difference between the utility values of two wealth values $(u_a(x_1) - u_a(x_2))/u_a(x_1)$ becomes smaller and this could pose a challenge to the optimization of ODRA and OIRA when the improvement of utility value is numerically very small. To resolve this issue, we apply a positive affine transformation to the utility values as $u^*_a(x) = (u_a(x) - u_a(1))/(u_a(1.1) - u_a(1))$ for all x and use the transformed utility values $u^*_a(x)$ in the loss function during training of ODRA and OIRA. This helps the optimization process and at the same time does not alter the problem formulation of the optimization as utility functions u_a and u^*_a represent the same set of underlying preferences.

4.4 Liquidity Provision Strategies

Below we outline the main strategies we compare in different regimes:

1. Optimal static allocation (OSA): This strategy computes $\mathbf{x}$ that optimizes the value of $u_a(V(\mathbf{x} \mid \mathbf{P}_1, \ldots, \mathbf{P}_N))$ from Section 3.1. This is the only strategy that does not explicitly use liquidity reallocations (though it can be seen as a τ-reset strategy with $\tau > T$), and it coincides with the work of [14].
2. Uniform liquidity τ-reset allocation (ULRA): For a fixed τ, this strategy mints an equal μ units of liquidity for each of the $2\tau + 1$ contiguous buckets considered in a reallocation. τ is chosen to be as large as possible so the LP makes use of the entire wealth at their disposal at a reset to reallocate liquidity.

[4] In the appendix of the full version of the paper we also provide a natural variant of OIRA for LPs exhibiting risk-aversion via logarithmic utilities as in [11]. For this model we provide convex optimization methods to solve for optimal allocations.

[5] The codebase we use to run experiments is available open-source at `https://github.com/Evensgn/uniswap-active-lp`.

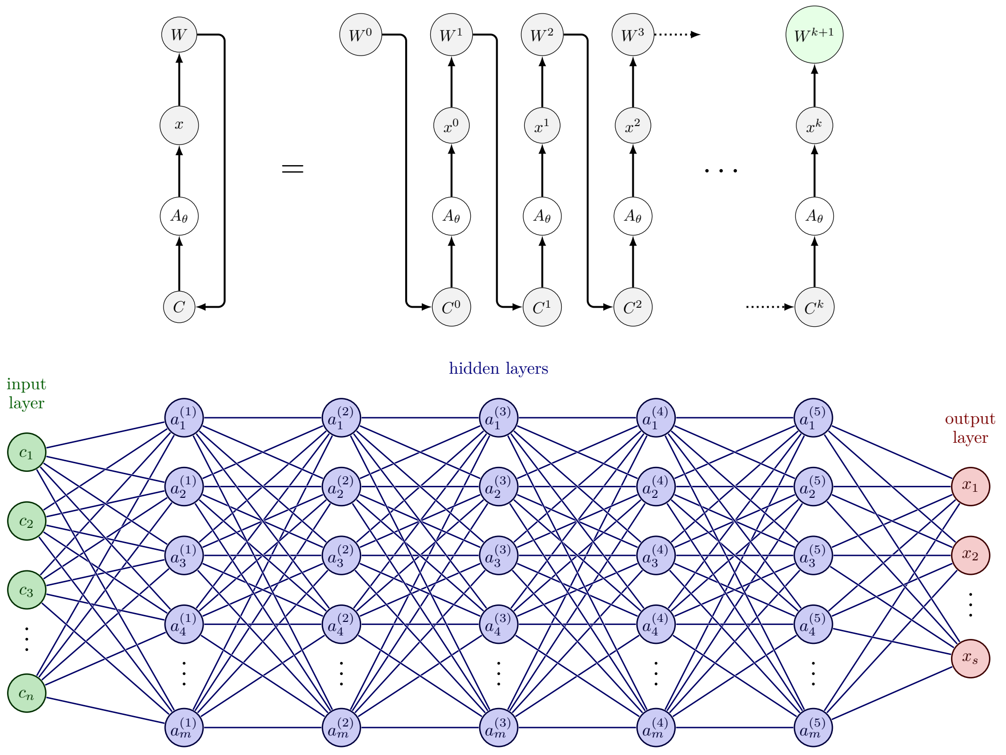


Figure 5 The top image provides a visualization of the recurrence in A_θ for the overall objective $\hat{V}^u_\tau(A_\theta \mid \mathbf{P}_1, \ldots, \mathbf{P}_N)$ which we exploit to compute gradients in a similar fashion to recurrent neural networks. In this image, C denotes the context that is fed to the neural network A_θ as features. A relevant feature at each epoch is the wealth that the LP has accumulated at the beginning of the epoch W^i, which is exemplified via an arrow in the figure. The overall objective is the given utility function applied to the wealth at the end of the final epoch W^{k+1}. The bottom image provides a visualization of the neural network architecture we use for A_θ. There is a soft-max function applied to the output layer. The recurrent structure of the objective's dependence with respect to the NN parameters, $\theta \in \boldsymbol{\theta}$ allow us to use techniques from recurrent neural networks to compute the gradient of the objective $u(W^{k+1})$ with respect to θ.

3. Uniform proportional τ-reset allocation (UPRA): For a fixed τ, this allocates wealth in equal proportions to each of the $2\tau + 1$ buckets after a reset (in general this does not result in a uniform liquidity allocation as the cost per unit of liquidity in each bucket may be different).
4. Optimal context-independent τ-reset allocation (OIRA): For a fixed τ, this computes the optimal single allocation vector to be used at every reset. In other words, the LP computes an optimal $(A_{-\tau}, \ldots, A_\tau)$, to be used to allocate liquidity around the reference bucket at each reallocation.
5. Optimal context-dependent τ-reset allocation (ODRA): For fixed τ, this is solved with the Neural Network formulation of Section 4.3.

5 Experimental Setup: Contract-Market Prices

In this section, we describe a family of empirically-informed contact-market price sequences against which we will optimize τ-reset strategies and we use historical data to inform this stochastic price model.

5.1 Modeling Contract-Market Prices

For this, we use a similar approach to [14], which is in turn inspired by [10], to provide a family of liquidity-independent contract-market price distributions. This makes use of an external stochastic process to define a sequence of market prices, together with non-arbitrage trades that affect the contract price and arbitrage trades that act to bring contract prices closer to market prices.

We assume that contract-market prices are generated over each of $R > 0$ *rounds*. At the beginning of each round, market prices change randomly according to a stochastic process $\mathcal{P}_M$. During the r-th round, after the contract-market price update, there are some number, $k_r \geq 0$, of non-arbitrage trades which impact contract price, P_c, only. Each non-arbitrage trade is either a purchase or a sale, this determined uniformly at random with probability 1/2. The effect of such a trade is that the contract price changes to $(1+\lambda_r)P_c$ or $(1+\lambda_r)^{-1}P_c$ respectively, where $\lambda_r > 0$, depending on whether a purchase or sale occured. Crucially, trades are price-based in our model rather than volume based, which in turn ensures that contract-market prices evolve independent of liquidity provided by an LP. It is precisely this exogenous uncertainty to LP actions that will allows us to compute optimal τ-reset strategies via the methods of Section 4, as this allows us to sample price paths first and then optimize LP allocation functions.

We also model arbitrage trades whose role is to bring contract prices close to market prices. For this, we follow [14], and with a Uniswap contract fee rate, $\gamma \in (0,1)$, we let $I_\gamma(P_m) = [(1-\gamma)P_m, (1-\gamma)^{-1}P_m]$ be the *no-arbitrage interval* around the market price P_m. If the contract price exits this no-arbitrage interval, we assume a arbitrage trade brings the contract price to the closest price in the interval. That is, if $P_c < (1-\gamma)P_m$ we assume that arbitrage trade moves the contract price to $(1-\gamma)P_m$, and if $P_c > (1-\gamma)^{-1}P_m$ we assume that arbitrage trade moves the contract price to $(1-\gamma)^{-1}P_m$.

▶ **Definition 11** (Round-Based Liquidity-Independent Price Distribution). *We say that $\mathcal{P}$ is a* round-based liquidity-independent price distribution *when it is a distribution that is parameterized by:*

- *$R > 0$: the number of rounds,*
- *$\gamma \in (0,1)$: the fee rate of the Uniswap contract,*
- *$\mathcal{P}_M$: the stochastic process governing market price updates at the beginning of each round,*
- *$\mathbf{k} = (k_r)_{r=1}^R$ with $k_r > 0$: the number of non-arbitrage trades in each round $r \in \{1,\dots,R\}$, and*
- *$\boldsymbol{\lambda} = (\lambda_r)_{r=1}^R$ with $\lambda_r > 0$: the multiplicative impact of a non-arbitrage trade on contract price for each round $r \in \{1,\dots,R\}$.*

When we wish to specify the resulting round-based price distribution, we write this as $\mathcal{P}(R,\gamma,\mathcal{P}_M,\mathbf{k},\boldsymbol{\lambda})$.

We model $\mathcal{P}_M$ as a geometric Brownian motion with parameters estimated from historical price data between token pairs. We also explore multiple regimes of time-varying non-arbitrage trade by varying $\boldsymbol{\lambda}$ (the framework is flexible enough to permit arbitrary values of λ_r for each round).

5.2 Market Prices as a Geometric Brownian Motion

We model the stochastic nature of market prices, $\mathcal{P}_M$, as a Geometric Brownian Motion (GBM). If the time series is given by $X_1,\dots,X_T$, then the successive multiplicative increments of the time series are i.i.d lognormally distributed. If we let $Z_i = \log\left(\frac{X_i}{X_{i-1}}\right)$, then $Z_2,\dots Z_T \sim$

iid $\mathcal{N}(\mu, \sigma^2)$ with *drift*, μ, and *diffusion*, σ. We estimate these parameters on per-minute time series data for ETH/BTC prices (the *low volatility regime*) and ETH/USDT (the *high volatility regime*) from March 2022 through February 2023. For each time series, we estimate the drift and diffusion via standard MLE methods. The estimates obtained were $(\hat{\mu}, \hat{\sigma}^2) = (4.835 \times 10^{-8}, 1.946 \times 10^{-7})$ and $(\hat{\mu}, \hat{\sigma}^2) = (-1.140 \times 10^{-6}, 8.329 \times 10^{-7})$ for the low and high volatility regimes respectively.

5.3 Contract Price Updates

Whereas arbitrage trades are specified by the fee rate of the contract, non-arbitrage trades are parametrized by $\mathbf{k} = (k_r)_{r=1}^R$ and $\boldsymbol{\lambda} = (\lambda_r)_{r=1}^R$, which specify the number and multiplicative magnitude of price change updates arising from non-arbitrage trades in a given round. In our experiments, we fix $k_r = 10$ for each round and introduce time-varying non-arbitrage price flow by explicitly modulating $\boldsymbol{\lambda}$ before sampling from $\mathcal{P}$. In particular we explore $\boldsymbol{\lambda}$ such that $\lambda_r = \bar{\lambda} + \alpha \cdot tanh(10(t/T - 0.5))$, where $\bar{\lambda} > 0$ is the average λ_r value over the time horizon and $\alpha > 0$ is the variation exhibited in λ_r around the mean.

6 Experimental Results

In this section, we explore the increase in earnings that LPs can gain through the use of dynamic allocation strategies, studying the performance of various liquidity provision strategies in a multitude of economic environments modulated by contract/market price volatility, LP risk-aversion, and reallocation costs. Most importantly, we find many settings in which optimal τ-reset strategies outperform simpler liquidity provision strategies. In all the experiments that follow, we assume a default setting of $(W, \gamma, R, k_r, \bar{\lambda}, \alpha, \eta) = (1, 0.003, 1000, 10, 0.00005, 0.00005, 0.01)$. When deviating from the default setting we clarify which parameters are changed. In addition, we assume that the buckets of the v3 contract $\boldsymbol{\mu} = \{B_{-m}, .., B_n\}$ are given by $B_i = [a_i, b_i] = [\phi^i, \phi^{i+1}]$ for $\phi = 1.0001^{10}$.

6.1 The Impact of Price Volatility

In Figures 6 and 7 we plot the performance of all LP strategies as we modulate $\mathcal{P}_M$ from low to high volatility as well as the risk-aversion of the LP. Figure 6 focuses on only comparing OIRA, ODRA and OSA to tease out the relative performance of ODRA vs. OIRA. Figure 7 incorporates UPRA and ULRA, from where we can see that their performance is almost identical. The NN-based ODRA outperforms all strategies, especially OSA which does not make use of reallocations. As risk-aversion increases, we see that the distinction between ODRA and OIRA becomes more clear in the plots, however, this does not imply a greater magnitude of performance due to the fact that different risk-aversion values give rise to different scales. In addition, we see that OIRA generally exhibits optimal performance with $\tau > 1$ whereas all other τ-reset strategies in this setting perform better with $\tau = 1$.

We see that for lower τ values, higher $\mathcal{P}_M$ leads to a greater separation between ODRA and OIRA in performance. Moreover, Figure 8 plots allocation profiles for ODRA and OIRA as we modulate risk-aversion and $\mathcal{P}_M$. As expected, with higher risk-aversion we see a larger spread in allocations, as LPs may seek to decrease the variance in their earnings with wider positions. On the other hand, as $\mathcal{P}_M$ increases in volatility, we see that LP positions for both ODRA and OIRA become narrower. This is likely due to the fact that the expected number of reallocations is higher in the high volatility setting than in the low volatility setting for the same τ. For a lower frequency of reallocations, the allocated liquidity is used for longer time periods, hence an LP may wish to spread liquidity over various buckets.

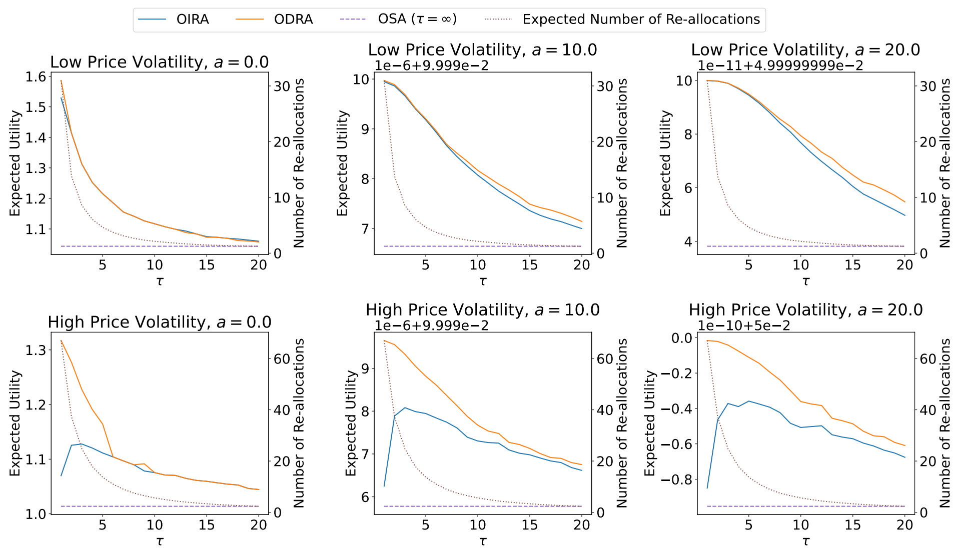


Figure 6 The performance of OIRA, ODRA and OSA strategies as we modulate both risk-aversion and $\mathcal{P}_M$. For each strategy, we plot the expected utility it achieves as a function of τ, and we also plot the expected number of reallocations that occur as a function of τ. The top row corresponds to a low volatility $\mathcal{P}_M$, empirically informed from ETH/BTC prices, and the bottom row corresponds to high volatility $\mathcal{P}_M$, empirically informed from ETH/USDC prices. The columns correspond to risk-aversion values $a = 0, 10$, and 20, respectively from left to right. When scientific notation is used for the y-axis values in certain subplots, it is denoted by a number above the respective y-axis.

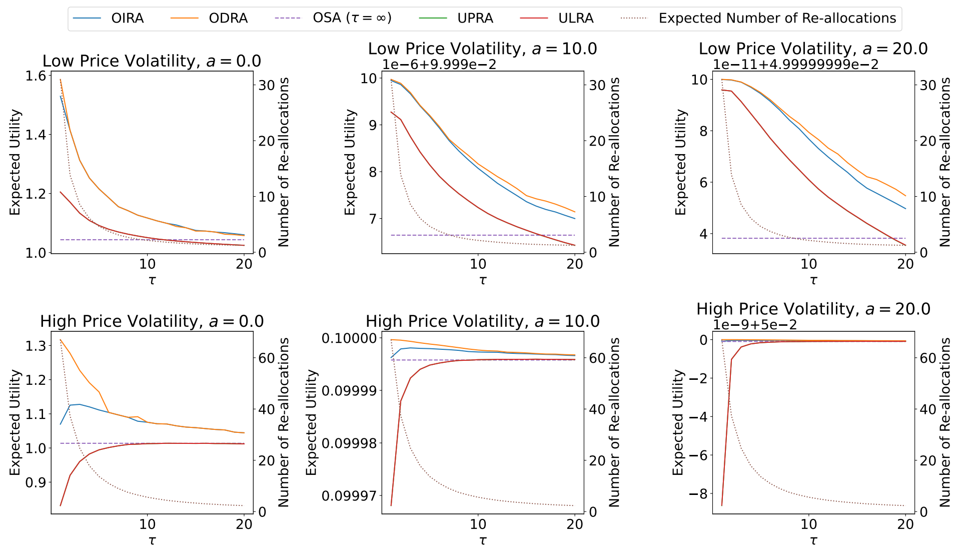


Figure 7 The performance of all strategies as we modulate both risk-aversion and $\mathcal{P}_M$. For each strategy we plot the expected utility it achieves as a function of τ, and we also plot the expected number of reallocations that occur as a function of τ. The top row corresponds to a low volatility $\mathcal{P}_M$, empirically informed from ETH/BTC prices, and the bottom row corresponds to high volatility $\mathcal{P}_M$, empirically informed from ETH/USDC prices. The columns correspond to risk-aversion values $a = 0, 10$, and 20, respectively from left to right. When scientific notation is used for the y-axis values in certain subplots, it is denoted by a number above the respective y-axis.

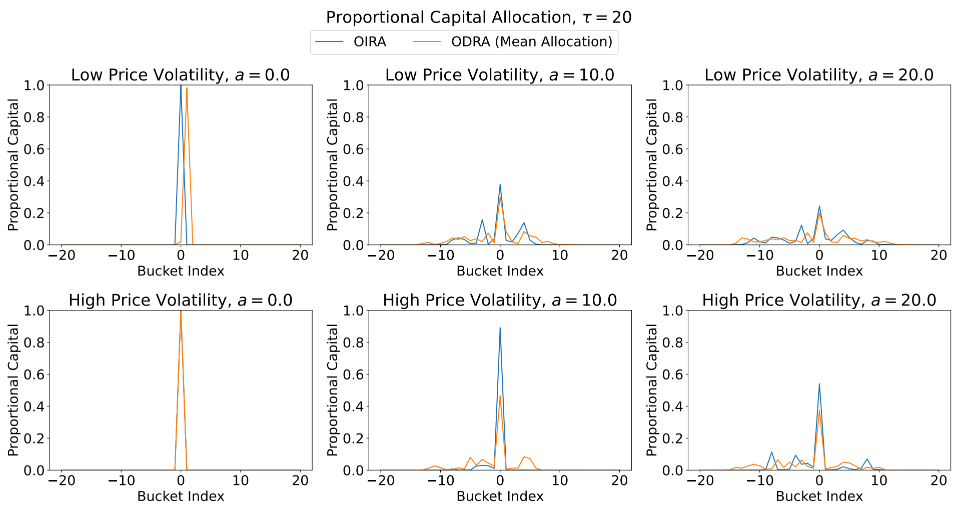


Figure 8 OIRA allocation and ODRA average allocations for $\tau = 20$ as we modulate both risk-aversion and $\mathcal{P}_M$. The top row corresponds to a low volatility $\mathcal{P}_M$, empirically informed from ETH/BTC prices, and the bottom row corresponds to high volatility $\mathcal{P}_M$, empirically informed from ETH/USDC prices. The columns correspond to risk-aversion values $a = 0, 10$, and 20, respectively from left to right.

6.2 Varying Non-arbitrage Flow

In Figure 9 we modulate the volatility of $\mathcal{P}_M$ and the magnitude of non-arbitrage flow in $\boldsymbol{\lambda}$ by modulating α, the amplitude of change in $\boldsymbol{\lambda}$ while keeping mean $\boldsymbol{\lambda}$ the same. The most salient observation is that as $\boldsymbol{\lambda}$ becomes more time-varying, the NN-based approach of ODRA increases its performance relative to OIRA. This is to be expected due to the fact that ODRA can incorporate temporal context in deciding an allocation after a reset, and the non-arbitrage flow inherently has the temporal context of increased importance as α increases.

In Figure 10 and 11 we also modulate $\boldsymbol{\lambda}$ albeit by jointly modulating amplitude(α) and mean of λ_r values, $\bar{\lambda}$. Once more we see that the NN-based ODRA strategy outperforms all strategies, and we see that the optimal τ values for ODRA drastically differ in the high volatility $\mathcal{P}_M$ over those of Figure 9. Moreover in Figure 11 we see that both increased non-arbitrage flow and $\mathcal{P}_M$ volatility contribute to more spread allocations. LPs make profits from non-arbitrage trades, hence they stand to obtain more fees with wider positions for larger flows of non-arbitrage trade.

6.3 The Impact of Risk-aversion

As mentioned in the previous sections, risk-aversion mostly impacts the allocations used in ODRA and OIRA LP strategies. In Figure 12 we make fine-grained modulations of risk-aversion and see that indeed LPs spread their liquidity more as they become more risk-averse. A larger spread of liquidity allocation typically implies lower expected earnings for an LP as they have less proportional liquidity at prices that are traded at, but at the same time, there is less risk of missing out on fees due to prices escaping their position or suffering impermanent loss due to price deviating from the initial price.

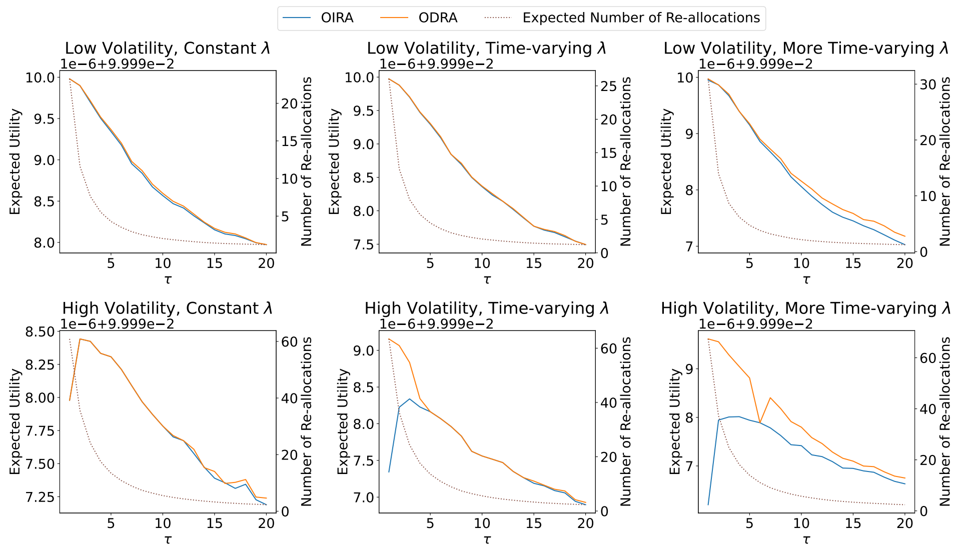


Figure 9 The performance of OIRA and ODRA strategies as we modulate $\mathcal{P}_M$ and $\boldsymbol{\lambda}$. For all plots, we use $a = 10$ for risk aversion and $\bar{\lambda} = 0.00005$ and we modulate the α in $\{0.0, 0.00003, 0.00005\}$ in columns from left to right. The top row plots low volatility $\mathcal{P}_M$ and the bottom row plots high volatility $\mathcal{P}_M$. When scientific notation is used for the y-axis values in certain subplots, it is denoted by a number above the respective y-axis.

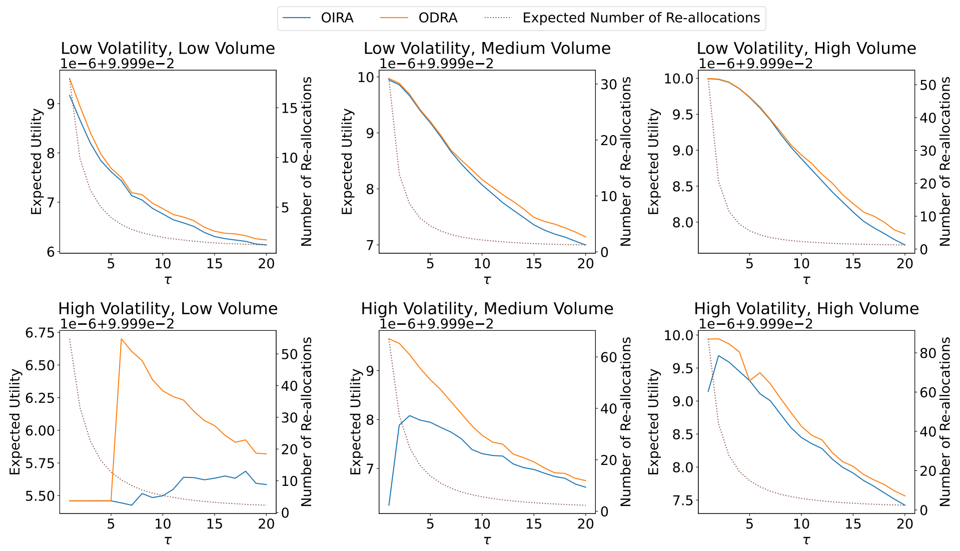


Figure 10 The performance of OIRA and ODRA strategies as we modulate $\mathcal{P}_M$ and $\boldsymbol{\lambda}$. For all plots, we use $a = 10$ for risk aversion and we modulate the $(\bar{\lambda}, \alpha) \in \{(0.000025, 0.000025), (0.00005, 0.00005), (0.000075, 0.000075)\}$ in columns from left to right. The top row plots low volatility $\mathcal{P}_M$ and the bottom row plots high volatility $\mathcal{P}_M$. When scientific notation is used for the y-axis values in certain subplots, it is denoted by a number above the respective y-axis.

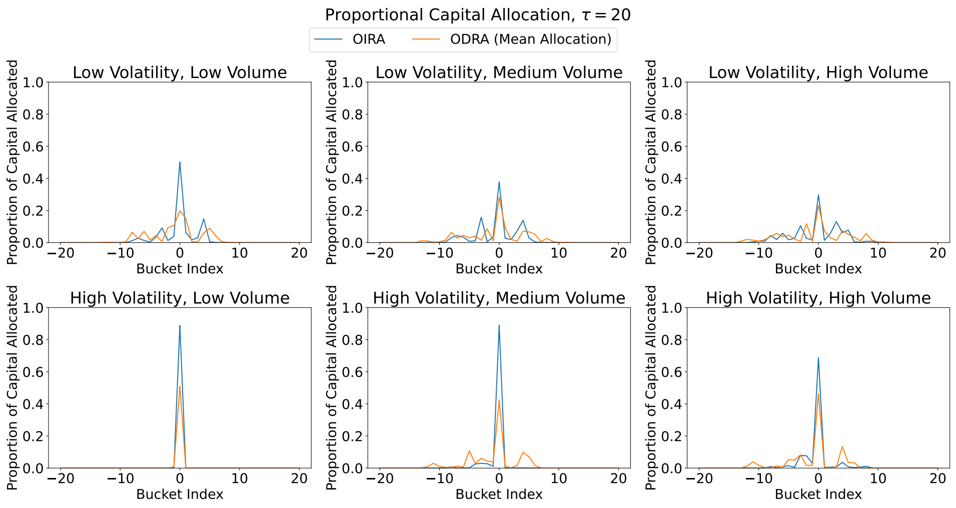


Figure 11 OIRA allocation and ODRA average allocation as we modulate $\mathcal{P}_M$ and $\boldsymbol{\lambda}$. For all plots, we use $a = 10$ for risk aversion and we modulate the $(\bar{\lambda}, \alpha) \in \{(0.000025, 0.000025), (0.00005, 0.00005), (0.000075, 0.000075)\}$ in columns from left to right. The top row plots low volatility $\mathcal{P}_M$ and the bottom row plots high volatility $\mathcal{P}_M$.

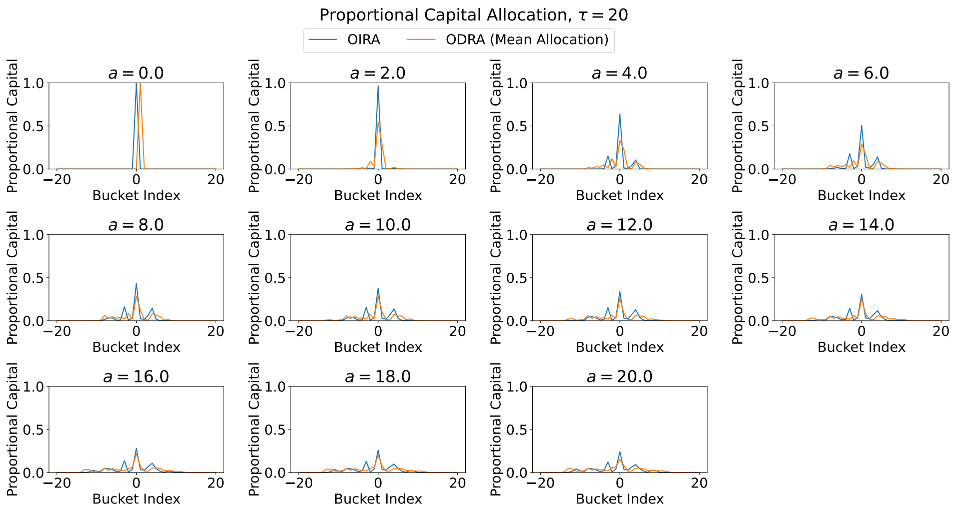


Figure 12 OIRA allocation and ODRA average allocation for $\tau = 20$ for low volatility $\mathcal{P}_M$ as we modulate risk-aversion from $a = 0$ to $a = 20$.

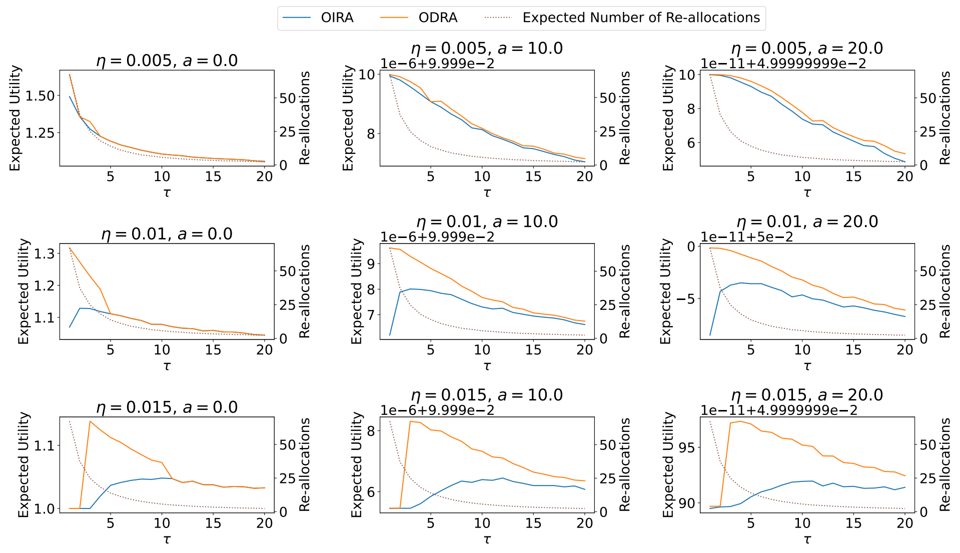


Figure 13 The performance of OIRA and ODRA strategies as we modulate η and risk-aversion. For all plots, we use high volatility $\mathcal{P}_M$. We modulate a in $\{0, 10, 20\}$ in columns from left to right and $\eta \in \{0.005, 0.01, 0.015\}$ in rows from top to bottom. When scientific notation is used for the y-axis values in certain subplots, it is denoted by a number above the respective y-axis.

6.4 The Impact of Reallocation Costs

In Figure 13 we modulate the cost of reallocation, η. We see that higher η values lead to higher optimal τ values for both OIRA and ODRA strategies. This is to be expected, for although low τ values might lead to higher gains in fees, this also leads to more frequent resets which in turn come with a higher cost.

7 Conclusion

This paper fills existing gaps in the literature regarding strategic liquidity provision strategies for LPs in Uniswap v3. Whereas earlier important work has either optimized for complex liquidity positions in static environments, or simple positions with dynamic reallocations, our work simultaneously provides complex, context-dependent liquidity allocations that dynamically reallocate as prices evolve in v3 contracts. Our results show that such liquidity provision strategies provide large gains for LPs in multiple economic environments for decentralized exchanges. Natural directions of future work include: incorporating LVR [20] into the objective definition to optimize the strategy of a delta-hedged LP, incorporating a game-theoretic framework to liquidity provision which is more apt for large LPs and modelling competition between different pools such as v2 and v3 pools for same token pairs.

References

1 Hayden Adams. Uniswap whitepaper, 2018. URL: `https://hackmd.io/@HaydenAdams/HJ9jLsfTz`.

2 Hayden Adams, Noah Zinsmeister, and Dan Robinson. Uniswap v2 core, 2020. URL: `https://uniswap.org/whitepaper.pdf`.

3 Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core, 2021. URL: `https://uniswap.org/whitepaper-v3.pdf`.

4 Guillermo Angeris and Tarun Chitra. Improved price oracles: Constant function market makers. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 80–91, 2020.

5 Guillermo Angeris, Alex Evans, and Tarun Chitra. When does the tail wag the dog? Curvature and market making. *arXiv preprint arXiv:2012.08040*, 2020.

6 Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, and Tarun Chitra. An analysis of Uniswap markets. *arXiv preprint arXiv:1911.03380*, 2019.

7 Jun Aoyagi. Lazy liquidity in automated market making. *Available at SSRN 3674178*, 2020.

8 Kenneth Joseph Arrow. *Aspects of the theory of risk-bearing.* Helsinki, 1965.

9 Yogev Bar-On and Yishay Mansour. Uniswap liquidity provision: An online learning approach. *arXiv preprint arXiv:2302.00610*, 2023.

10 Agostino Capponi and Ruizhe Jia. The adoption of blockchain-based decentralized exchanges. *arXiv preprint arXiv:2103.08842*, 2021.

11 Álvaro Cartea, Fayçal Drissi, and Marcello Monga. Decentralised finance and automated market making: Predictable loss and optimal liquidity provision. *Available at SSRN 4273989*, 2022.

12 Alex Evans. Liquidity provider returns in geometric mean markets. *arXiv preprint arXiv:2006.08806*, 2020.

13 Alex Evans, Guillermo Angeris, and Tarun Chitra. Optimal fees for geometric mean market makers. *arXiv preprint arXiv:2104.00446*, 2021.

14 Zhou Fan, Francisco J. Marmolejo Cossío, Ben Altschuler, He Sun, Xintong Wang, and David C. Parkes. Differential liquidity provision in Uniswap v3 and implications for contract design. In *3rd ACM International Conference on AI in Finance, ICAIF*, pages 9–17, 2022.

15 Rafael Frongillo, Maneesha Papireddygari, and Bo Waggoner. An axiomatic characterization of CFMMs and equivalence to prediction markets. *arXiv preprint arXiv:2302.00196*, 2023.

16 Mohak Goyal, Geoffrey Ramseyer, Ashish Goel, and David Mazières. Finding the right curve: Optimal design of constant function market makers. *arXiv preprint arXiv:2212.03340*, 2022.

17 Lioba Heimbach, Eric Schertenleib, and Roger Wattenhofer. Risks and returns of Uniswap v3 liquidity providers. *arXiv preprint arXiv:2205.08904*, 2022.

18 Max. Introducing Alpha Vaults–an LP strategy for Uniswap V3, 2021. URL: `https://medium.com/charmfinance/introducing-alpha-vaults-an-lp-strategy-for-uniswap-v3-ebf500b67796`.

19 Jason Milionis, Ciamac C Moallemi, and Tim Roughgarden. A Myersonian framework for optimal liquidity provision in automated market makers. *arXiv preprint arXiv:2303.00208*, 2023.

20 Jason Milionis, Ciamac C Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Automated market making and loss-versus-rebalancing. *arXiv preprint arXiv:2208.06046*, 2022.

21 Michael Neuder, Rithvik Rao, Daniel J Moroz, and David C Parkes. Strategic liquidity provision in uniswap v3. *arXiv preprint arXiv:2106.12033*, 2021.

22 John W Pratt. Risk aversion in the small and in the large. *Econometrica*, 32:122–136, 1964.

23 Jan Christoph Schlegel, Mateusz Kwaśnicki, and Akaki Mamageishvili. Axioms for constant function market makers. *Available at SSRN*, 2022.

24 Martin Tassy and David White. Growth rate of a liquidity provider's wealth in $xy = c$ automated market makers, 2020.

25 Dave White, Martin Tassy, Charlie Noyes, and Dan Robinson. Uniswap's financial alchemy, 2020. URL: `https://research.paradigm.xyz/uniswaps-alchemy`.

26 Wenqi Zhao, Hui Li, and Yuming Yuan. Understand volatility of Algorithmic Stablecoin: Modeling, verification and empirical analysis. In *Financial Cryptography and Data Security*, volume 12676 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 2021.

Post-Quantum Single Secret Leader Election (SSLE) from Publicly Re-Randomizable Commitments

Dan Boneh
Stanford University, CA, USA

Aditi Partap
Stanford University, CA, USA

Lior Rotem
Stanford University, CA, USA

Abstract

A *Single Secret Leader Election (SSLE)* enables a group of parties to randomly choose exactly one leader from the group with the restriction that the identity of the leader will be known to the chosen leader and nobody else. At a later time, the elected leader should be able to publicly reveal her identity and prove that she is the elected leader. The election process itself should work properly even if many registered users are passive and do not send any messages. SSLE is used to strengthen the security of proof-of-stake consensus protocols by ensuring that the identity of the block proposer remains unknown until the proposer publishes a block. Boneh, Eskandarian, Hanzlik, and Greco (AFT'20) defined the concept of an SSLE and gave several constructions. Their most efficient construction is based on the difficulty of the Decision Diffie-Hellman problem in a cyclic group.

In this work we construct the first efficient SSLE protocols based on the standard Learning With Errors (LWE) problem on integer lattices, as well as the Ring-LWE problem. Both are believed to be post-quantum secure. Our constructions generalize the paradigm of Boneh et al. by introducing the concept of a re-randomizable commitment (RRC). We then construct several post-quantum RRC schemes from lattice assumptions and prove the security of the derived SSLE protocols. Constructing a lattice-based RRC scheme is non-trivial, and may be of independent interest.

2012 ACM Subject Classification Security and privacy → Cryptography

Keywords and phrases Consensus, Leader Election, Post-Quantum, Lattice Cryptography, Blockchain

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.26

Related Version *Full Version*: `https://eprint.iacr.org/2023/1241.pdf`

Funding This work was funded by NSF, DARPA, the Simons Foundation, UBRI, and NTT Research. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

1 Introduction

Leader election is a core component of many consensus protocols used in practice. In proof-of-work systems such as [34], the identity of the leader remains hidden until the moment that the leader publishes a proposed block. In contrast, in many proof-of-stake systems, the identity of the leader is known in advance, long before the leader publishes a proposed block. This opens up the leader to certain attacks, including denial of service, that may prevent the chosen leader from publishing the newly created block. This in turn, can lead to a liveness failure for the chain.

In response, several works have studied *secret* leader election, where the identity of a randomly chosen leader remains secret until she publishes the new block and reveals herself as the leader [25, 31, 27, 8]. The added secrecy protects the leader from attacks that may prevent her from publishing the new block. However, existing proposals for secret leader election work by electing a few potential leaders *in expectation*, and describing a run-off procedure so that exactly one of the potential leaders is recognized as the final leader once all potential leaders have revealed themselves. The possibility of several potential leaders, however, can lead to wasted effort and may even cause a safety violation in case of an attack on the run-off procedure.

This issue motivates the need for a different type of leader election protocol, called a *Single Secret Leader Election*, or SSLE [16] (see also [21]). An SSLE protocol is comprised of two phases.

- In the first phase, parties may register to participate in leader elections. This step involves publishing some public information on a public bulletin board, while keeping some secret information associated with it private.
- In the second phase, elections are held using a protocol that is executed by the participating parties. The election protocol uses a randomness beacon and the public information on the bulletin to choose a leader among the parties. At a later time, the leader can declare themselves as such by providing a proof that they were selected as the leader.

Informally, an SSLE protocol needs to satisfy three security properties. **Uniqueness** asserts that at most a single party can prove they were elected as leader. **Fairness** requires that all participating parties have the same probability of being elected as leader, even if some parties are malicious. **Unpredictability** means that until the leader reveals itself, its identity should remain essentially hidden from the other parties, even if a subset of them colludes. It was recently shown that relying on SSLE leads to more efficient consensus protocols than relying on a secret leader election protocol that elects few leaders in expectation [7].

The concept of SSLE was formalized by Boneh, Eskandarian, Hanzlik, and Greco [16] who also presented a number of constructions. Their most efficient construction is based on the Decision Diffie-Hellman problem (DDH) in cyclic groups. We refer to this SSLE protocol as the **BEHG protocol**. The Ethereum Foundation optimized BEHG to obtain *Whisk* [30], which is the current proposal for SSLE to be used in Ethereum consensus. Since then, additional works have suggested alternative SSLE constructions with various security and efficiency tradeoffs (see, for example, [23, 40, 18, 9, 19, 24]).

Due to the potential long-term risk of a large scale quantum computer [41] there is a desire to also develop a *post-quantum* secure SSLE. One approach, already in [16], is an SSLE protocol based on fully homomorphic encryption (FHE). A further optimized FHE-based construction was recently proposed by Freitas et al. [24]. However, the complexity of these proposals is far greater than the simple DDH-based scheme. Another elegant approach to post-quantum SSLE was proposed by Sanso [40], who showed how to adapt Whisk to use an isogeny-based assumption, which is believed to be post-quantum secure. Finally, Drake [23] proposed an SSLE protocol that can be made post-quantum secure, but the proposal inherently relies on the availability of an anonymous broadcast channel (e.g., ToR).

Our results. In this paper we construct the first practical post-quantum SSLE protocols based on the Learning With Errors (LWE) problem [38] and Ring-LWE problem [33]. We do so by generalizing the BEHG protocol using a new concept we call a re-randomizable commitment (RRC). We show that an RRC together with a shuffle protocol gives an SSLE. We then construct a number of RRC schemes from lattices. The next section gives a detailed overview of the construction and explains the technical challenges in building an RRC from lattices.

1.1 Technical Overview

We briefly sketch the main ideas behind our construction. We begin with an abstract view of the BEHG protocol. Then, we present the notion of re-randomizable commitments (RRC) used by this protocol. Finally, we present our new lattice-based post-quantum RRCs for instantiating the abstract BEHG protocol.

The BEHG approach. The BEHG protocol employs a commit-and-shuffle approach. The following is a generalized and abstract view of the protocol.

- When party i registers for elections, it chooses some secret key k_i, computes a commitment c_i to k_i, and publishes c_i. We will define what is needed of this commitment in a minute. To avoid duplicity of secrets, each party also publishes a deterministic hash of k_i.
- At election time, participating parties run a protocol to shuffle and rerandomize the commitments. For simplicity of presentation in this overview, let us assume that the shuffle protocol works as follows: in each round, one of the parties locally permutes the entire list of commitments and then rerandomizes each of the commitments. It then publishes the new list of commitments, and proves in zero-knowledge that this new list is well-formed (i.e., it is obtained from the previous list by permuting and rerandomizing the commitments). Once the shuffle protocol is done, the parties obtain a list of commitments $\tilde{c}_1, \ldots, \tilde{c}_n$, where each $\tilde{c}_i$ is a rerandomization of $c_{\pi(i)}$ for some unknown permutation π on $\{1, \ldots, n\}$. They then let the randomness beacon choose an index $i^* \xleftarrow{\$} \{1, \ldots, n\}$, and party $j^* = \pi(i^*)$ is the chosen leader. In due time, party j^* can prove that it was elected by publishing k_{j^*} and the other parties can check this value against the commitment $\tilde{c}_{i^*}$.

Re-randomizable commitments. We identify several properties that the commitment scheme being used must satisfy for the resulting SSLE protocol to be correct and secure. First, the commitments have to be **re-randomizable** in a very specific sense. Given a commitment c to some value k, one should be able to re-randomize c without knowledge of k or the randomness used to generate c. Moreover, given a value k and a (potentially re-randomized) commitment $\tilde{c}$, one should be able to efficiently test whether $\tilde{c}$ is a commitment to k. In particular, this test should not require the randomness used for re-randomization. In the BEHG protocol, this means that the original committer to $\tilde{c}_{i^*}$ can: (i) recognize itself as the winner of the elections (by checking if $\tilde{c}_{i^*}$ is a commitment to k_{j^*}); and (ii) prove that it won by publishing k_{j^*}.

The commitment scheme should also satisfy the standard notion of **binding**. This means that it should be infeasible to produce a commitment c alongside two *distinct* values k and k', such that c passes both as a commitment to k and as a commitment to k'. In the context of the BEHG protocol, this means that there is only a single party that can prove ownership of the chosen commitment $\tilde{c}_{i^*}$ by publishing k_{j^*}.

Finally, commitments should also be **unlinkable**. This means that given two commitments c_0 and c_1 to two random values, and a re-randomization $\tilde{c}$ for one of them, it should be infeasible to determine if $\tilde{c}$ is a re-randomization of c_0 or of c_1. This is essential for the BEHG protocol to achieve unpredictability: an adversary should not be able to link the chosen commitment $\tilde{c}_{i^*}$ to the original commitment c_{j^*} and therefore identify party j^* as the leader. Looking ahead, the use of re-randomizable commitments in the generalized BEHG SSLE protocol actually requires a stronger notion of unlinkability. We postpone the discussion on this matter and will revisit it shortly.

The DDH-based construction of re-randomizable commitments (RRCs) suggested by BEHG is as follows. Let $\mathbb{G}$ be a cyclic group of order p generated by $g \in \mathbb{G}$. A commitment c to a random value $k \xleftarrow{\$} \mathbb{Z}_p$ is a pair (g^r, g^{rk}) where $r \xleftarrow{\$} \mathbb{Z}_p$. To check if a commitment

$c = (c_1, c_2)$ is a consistent with a value k, once can simply check if $c_2 = c_1^k$. To re-randomize, one chooses a random $r' \xleftarrow{\$} \mathbb{Z}_p$ and outputs $\tilde{c} = (c_1^{r'}, c_2^{r'})$. The scheme is perfectly binding, and unlinkability easily follows from the DDH assumption.

It should be noted that previous works also considered other variants of re-randomizable commitments (see, for example, [5, 20]). However, in these works, opening a re-randomized commitment requires knowledge of the randomness used for re-randomization (or a function thereof). Such commitments are much simpler to construct, and indeed, many long-standing algebraic and lattice-based constructions can be easily re-randomized according to this weaker definition. Unfortunately, as discussed above, such commitments are insufficient for instantiating the BEHG protocol.

RRCs from LWE: A first attempt. Consider the following (flawed) RRC scheme. The secret key space is $\mathbb{Z}_q^n$, where q is a prime and $n \approx \lambda$ is the LWE hardness parameter. To commit to a random $\boldsymbol{k} \in \mathbb{Z}_q^n$ the *Commit* algorithm samples a uniformly random $\boldsymbol{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$ and outputs $(\boldsymbol{A}, \boldsymbol{u}) = (\boldsymbol{A}, \boldsymbol{A} \cdot \boldsymbol{k} + \boldsymbol{e})$, where $\boldsymbol{e}$ is an LWE noise vector and $m > n$. To test whether a key $\boldsymbol{k}$ is tied to a commitment $c = (\boldsymbol{A}, \boldsymbol{u})$, we can check whether $\boldsymbol{A} \cdot \boldsymbol{k}$ is close (say, in Euclidean distance) to $\boldsymbol{u}$. We accept $\boldsymbol{k}$ if this is the case and reject otherwise. If $\boldsymbol{A}$ is chosen randomly and $m \approx n \log n$ ($\boldsymbol{A}$ is a "tall" matrix), a standard argument shows that with high probability over the choice of $\boldsymbol{A}$, there are no $\boldsymbol{k}, \boldsymbol{k}'$ and $\boldsymbol{u}$ such that $\boldsymbol{A} \cdot \boldsymbol{k} \approx \boldsymbol{u}$ and $\boldsymbol{A} \cdot \boldsymbol{k}' \approx \boldsymbol{u}$.

To re-randomize, the rerandomization algorithm samples a low-norm m-by-m matrix $\boldsymbol{R}$ and computes $c' = (\boldsymbol{A}', \boldsymbol{u}') = (\boldsymbol{R} \cdot \boldsymbol{A}, \boldsymbol{R} \cdot \boldsymbol{u})$. Since $\boldsymbol{R}$ is of low norm $\boldsymbol{Re}$ may only be slightly longer than $\boldsymbol{e}$. Hence, $\boldsymbol{Re}$ is also short and we have

$$\boldsymbol{A}' \cdot \boldsymbol{k} = \boldsymbol{R} \cdot \boldsymbol{A} \cdot \boldsymbol{k} \approx \boldsymbol{R} \cdot \boldsymbol{A} \cdot \boldsymbol{k} + \boldsymbol{R} \cdot \boldsymbol{e} \approx \boldsymbol{R} \cdot \boldsymbol{u} = \boldsymbol{u}'.$$

The noise does grow a bit with each re-randomization, which is why the scheme only supports a bounded number of re-randomizations (the LWE parameters can be chosen according to the number of re-randomizations required by the SSLE shuffle protocol). In terms of unlinkability, note that assuming LWE is hard, a fresh commitment $c = (\boldsymbol{A}, \boldsymbol{u})$ is just a pseudorandom matrix-vector pair. Moreover, if m is sufficiently greater than n and each row of $\boldsymbol{R}$ has high min-entropy, the leftover hash lemma [26, 29] shows that c' is also pseudorandom, which implies that the scheme is unlinkable.

The problem. Unfortunately, the above analysis is flawed. It is true that the scheme is binding when the matrix $\boldsymbol{A}$ is chosen uniformly at random from $\mathbb{Z}_q^{m \times n}$. But since $\boldsymbol{A}$ is part of the commitment c, the adversary may choose it from some other skewed distribution, thus breaking the binding argument. This is not just an issue of reworking the proof. The scheme is in fact insecure: fix any $\boldsymbol{k}$ and $\boldsymbol{k}'$ and it is easy to come up with a matrix $\boldsymbol{A}$ for which $\boldsymbol{A} \cdot \boldsymbol{k} \approx \boldsymbol{A} \cdot \boldsymbol{k}'$. To fix this issue, one might be tempted to choose the matrix $\boldsymbol{A}$ as part of the public parameters, or to force committers to choose $\boldsymbol{A}$ as the output of a hash function modeled as a random oracle. Indeed, this would make the scheme binding, but then it becomes unclear how to re-randomize the commitments.

The key observation. Let us revisit the naive "proof" of binding for the above construction. If $\boldsymbol{A}$ is indeed chosen uniformly at random, then with overwhelming probability there are no $\boldsymbol{k}$ and $\boldsymbol{k}'$ such that $\boldsymbol{A} \cdot \boldsymbol{k} \approx \boldsymbol{A} \cdot \boldsymbol{k}'$. In particular, this would suggest that for random $\boldsymbol{A}$, $\boldsymbol{k}$ and $\boldsymbol{k}'$ it holds that $\boldsymbol{A} \cdot \boldsymbol{k}$ and $\boldsymbol{A} \cdot \boldsymbol{k}'$ are almost surely far apart. Put differently, for a uniform $\boldsymbol{k}$ and $\boldsymbol{k}'$, there are *very few* matrices $\boldsymbol{A}$ for which $\boldsymbol{Ak} \approx \boldsymbol{Ak}'$. So what if instead of choosing a single $\boldsymbol{k}$, we make the *Commit* algorithm sample the commitment key k as a pair $(\boldsymbol{k}_1, \boldsymbol{k}_2)$ of independent and uniformly-random vectors? One could expect that for two such random pairs $(\boldsymbol{k}_1, \boldsymbol{k}_2)$ and $(\boldsymbol{k}'_1, \boldsymbol{k}'_2)$, the set of matrices $\boldsymbol{A}$ for which $\boldsymbol{A} \cdot \boldsymbol{k}_1 \approx \boldsymbol{A} \cdot \boldsymbol{k}'_1$ *and* $\boldsymbol{A} \cdot \boldsymbol{k}_2 \approx \boldsymbol{A} \cdot \boldsymbol{k}'_2$ is *even smaller*. Indeed, we show that for $\ell \approx n$, if one samples two ℓ-tuples $(\boldsymbol{k}_1, \ldots, \boldsymbol{k}_\ell)$ and $(\boldsymbol{k}'_1, \ldots, \boldsymbol{k}'_\ell)$ of vectors uniformly at random, then with very high probability a matrix $\boldsymbol{A}$ for which $\boldsymbol{A} \cdot \boldsymbol{k}_i \approx \boldsymbol{A} \cdot \boldsymbol{k}'_i$ for every i *simply does not exist.*

Alas, the proposed commitment scheme is binding for keys that are *random* tuples of vectors, but the binding security game allows the adversary to choose the "colliding" keys $(\boldsymbol{k}_1, \ldots, \boldsymbol{k}_\ell)$ and $(\boldsymbol{k}'_1, \ldots, \boldsymbol{k}'_\ell)$ as it pleases – they need not be uniformly random. On the face of it, it might seem that we are back to square one. Fortunately, this is not the case. The final observation is that for this construction, we *can* make the commitment algorithm choose the vectors $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_\ell$ as the output of a cryptographic hash function H, without hampering re-randomization. That is, to commit, one samples a matrix $\boldsymbol{A}$ and a key $k \xleftarrow{\$} \{0,1\}^\lambda$, computes $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_\ell \leftarrow \mathsf{H}(k)$ and outputs the commitment $c \leftarrow (\boldsymbol{A}, \{\boldsymbol{A} \cdot \boldsymbol{k}_i + \boldsymbol{e}_i\}_i)$ where all the $\boldsymbol{e}_i$s are independent LWE noise vectors. To test a key k against a commitment $c = (\boldsymbol{A}, \{\boldsymbol{u}_i\}_i)$, the *Test* algorithm simply recomputes $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_\ell$ from k and checks that $\boldsymbol{A} \cdot \boldsymbol{k}_i \approx \boldsymbol{u}_i$ for every $i = 1, \ldots, \ell$.

Adversarial re-randomizations. The construction that we just saw indeed satisfies the notion of unlinkability sketched above. Unfortunately, as we already mentioned, this notion is insufficient for the resulting SSLE protocol to achieve unpredictability. This reason is this: unlinkability only guarantees that if honestly-generated commitments $c_1, \ldots, c_n$ are honestly re-randomized and shuffled, an adversary cannot trace the re-randomized commitments to the original ones. In the SSLE protocol above, an honest re-randomization might follow an adversarial one. So we need to require unlinkability of commitments even after adversarial re-randomizations. We call this *strong unlinkability.*

In the DDH-based construction of BEHG strong unlinkability comes "for free". Unfortunately, this is not the case with our LWE-based RRC scheme. For example, consider an adversary that given a commitment $c = (\boldsymbol{A}, \{\boldsymbol{A} \cdot \boldsymbol{k}_i + \boldsymbol{e}_i\}_i)$, finds a matrix $\boldsymbol{R}$ such that $\boldsymbol{R} \cdot \boldsymbol{A}$ has short columns. The adversary then uses this $\boldsymbol{R}$ to re-randomize c into $\tilde{c} \leftarrow (\boldsymbol{R} \cdot \boldsymbol{A}, \{\boldsymbol{R} \cdot \boldsymbol{A} \cdot \boldsymbol{k}_i + \boldsymbol{e}_i\}_i)$. Now, even if we honestly re-randomize $\tilde{c}$, we will almost surely end up with a commitment $\hat{c}$ whose first coordinate is still a short-columns matrix. Hence, the adversary can easily trace $\hat{c}$ back to c.

We present several methods to thwart such attacks. In this overview, we focus on what we view as the simplest and most practical one. Ahead of time, all parties commit to the matrices $\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots$ they are going to use for re-randomization using a standard additively homomorphic commitment scheme. When a party now has to carry out its ith re-randomization, it does so using the matrix $\boldsymbol{R}_i + \boldsymbol{R}'_i$, where $\boldsymbol{R}'_i$ is a low norm matrix outputted by a public randomness beacon. Such a beacon can be external or implemented in various standard ways. Using the homomorphic properties of the commitment scheme, everyone can now compute a commitment to $\boldsymbol{R}_i + \boldsymbol{R}'_i$. The re-randomizer can hence prove that this is the matrix it used. Informally, since $\boldsymbol{R}_i$ was committed to ahead of time, it is independent of $\boldsymbol{R}'_i$. Hence, the re-randomizer is forced to use a high-entropy matrix for re-randomization, which guarantees the resulting commitment is from the appropriate distribution. Since $\boldsymbol{R}_i$ is always hidden, $\boldsymbol{R}_i + \boldsymbol{R}'_i$ has high min-entropy even given $\boldsymbol{R}'_i$, and we can still rely on the leftover hash lemma to argue that subsequent honest re-randomizations provide unlinkability.

Extending the scheme to Ring LWE. We extend our LWE-based RRC scheme to the ring setting, relying on the Ring Learning with Errors (Ring-LWE) assumption. As we discuss in Section 5 in detail, moving to the ring setting offers several gains in efficiency. Specifically, we work in a polynomial ring $\mathcal{R}$ modulo a cyclotomic polynomial f, which factors into a constant number of irreducible polynomials over $\mathbb{Z}_q$. Concretely, we choose $q = 3 \bmod 8$ so that f has exactly two irreducible factors f_1, f_2 over $\mathbb{Z}_q$ (but other choices are possible).

The construction follows the same template as our LWE-based construction, but the matrix $\boldsymbol{A}$ is now replaced with a vector of ring elements. To commit, one samples $\boldsymbol{a} \xleftarrow{\$} \mathcal{R}_q^m$, and a key $k \xleftarrow{\$} \{0,1\}^\lambda$, computes ℓ ring elements as $k_1, \dots, k_\ell \leftarrow \mathsf{H}(k)$ and the commitment is given by $c \leftarrow (\boldsymbol{a}, \{\boldsymbol{a} \cdot k_i + \boldsymbol{e}_i\}_i)$ where all $\boldsymbol{e}_i$s are independent RLWE noise vectors in $\mathcal{R}_q^m$. Re-randomization is done by sampling a low-norm matrix $\boldsymbol{R} \xleftarrow{\$} \mathcal{R}^{m \times m}$, and computing $c' = (\boldsymbol{R} \cdot \boldsymbol{a}, \{\boldsymbol{R} \cdot \boldsymbol{u}_i\}_i)$. To test a commitment $c = (\boldsymbol{a}, \boldsymbol{u})$ against a key k, one computes $k_1, \dots, k_\ell \leftarrow \mathsf{H}(k)$ and check that $\boldsymbol{a} \cdot k_i \approx \boldsymbol{u}_i$ for all i. Correctness and unlinkability are proven similarly to the integer case, with one exception: instead of relying on the leftover hash lemma, we rely on the regularity lemma of [42].

Two main observations make our ring-based scheme more efficient than our integer-based one:

- We can choose ℓ to be smaller than in the integer case, and still make the binding argument go through. Intuitively, the reason is that each entry of $\boldsymbol{a} \cdot k_i$ is now a polynomial in the ring $\mathcal{R}$ and not an integer. Thus, we may hope that it has more than $\log q$ bits of min-entropy (roughly the entropy of a random integer in $\mathbb{Z}_q$). If this is indeed the case, then the probability that $\boldsymbol{a} \cdot k \approx \boldsymbol{a} \cdot k'$, over the choice of random $\boldsymbol{a}, k, k'$, is much smaller than the probability that $\boldsymbol{a}^T \cdot \boldsymbol{k} \approx \boldsymbol{a}^T \cdot \boldsymbol{k}'$ in the integer case for random $\boldsymbol{a}, \boldsymbol{k}, \boldsymbol{k}' \xleftarrow{\$} \mathbb{Z}_q^n$. This would imply that we can choose ℓ to be smaller, resulting in smaller commitments. To argue that $\boldsymbol{a} \cdot k_i$ indeed has high min-entropy, we rely on the particular structure of the ring $\mathcal{R}$. If $k \neq k'$, it means that the polynomials must be distinct modulo f_1 or modulo f_2. Assume with loss of generality that they are distinct modulo f_1. Since f_1 is irreducible mod q, $a \cdot (k - k')$ is uniformly random in $\mathbb{Z}_q[x]/f_1$, and hence it has at least $\approx \mathsf{deg}(f_1) \cdot \log q$ bits of min entropy. This analysis is inspired by the statistically-binding commitments of Benhamouda, Krenn, Lyubashevsky, and Pietrzak [14].
- The second observation is that our use of the leftover hash lemma in the LWE setting incurred an overhead that can be avoided in the Ring LWE setting. To explain this point, we need to revisit our LWE unlinkability argument in more detail. Recall that we wanted to argue that if we have a commitment $c = (\boldsymbol{A}, \boldsymbol{U})$ and we re-randomize it to $c' = (\boldsymbol{R} \cdot \boldsymbol{A}, \boldsymbol{R} \cdot \boldsymbol{U})$, then the commitment c' we end up with is pseudorandom. The first step was to argue that c is pseudorandom, thanks to the LWE assumption. This step remains essentially unchanged here, relying on the Ring-LWE assumption instead. The second step was to rely on the leftover has lemma; this step required each row of $\boldsymbol{R}$ to have more than $\Omega((n + \ell) \cdot \log q)$ bits of min-entropy. This implied that m had to be set to be at least $(n + \ell) \cdot \log q$. In the ring setting, however, since each coordinate of $\boldsymbol{R}$ can have $\Omega(n)$ bits of min-entropy, m can be reduced to roughly $\log q$. This results in much "shorter" matrices $\boldsymbol{A}, \boldsymbol{U}$ making up the commitment.

Reducing communication. Catalano, Fiore, and Giuta [19] observed that when instantiating the BEHG protocol with a DDH-based RRC of the form $c = (g^r, g^{rk})$, the commitments of all parties can share the same first coordinate $h = g^r$, which is part of the public parameters. Then, to re-randomize N commitments $(h^r, g^{rk_1}, \dots, g^{rk_N})$, a shuffler can sample a single $r' \xleftarrow{\$} \mathbb{Z}_q$ and raise all the elements to the r'. This optimization cuts storage and communication by about half. It is tempting to implement this optimization using our lattice-based commitments; have all commitments share the first coordinate $\boldsymbol{A}$ (or $\boldsymbol{a}$ in the ring setting) and use a single re-randomization matrix $\boldsymbol{R}$ to re-randomize all commitments. The problem is that to retain unlinkability, the dimensions of $\boldsymbol{R}$ need to grow as a function of the number of commitments N, which may eliminate the gains of sharing $\boldsymbol{A}$ across all commitments. We discuss this further in the full version where we consider settings where this can still lead to some savings.

Post-quantum proof of shuffle. Recall that in the BEHG protocol, after each shuffle, the shuffling party has to prove that it indeed performed a valid shuffle; that is, it applied the *Randomize* algorithm of the RRC scheme to each commitment and then permuted the resulting commitments. This can be done by using any general-purpose non-interactive argument of knowledge, proving that the shuffler knows random coins for *Randomize* and a permutation that together yield the resulting list of re-randomized commitments (for such argument systems based on post-quantum secure assumptions, see for example [13, 11, 12, 4, 15, 28, 10, 6, 32, 2, 35] and the references therein).

When using our RLWE-based RRC commitments, we also show how we can change the recent lattice-based proof-of-shuffle protocol of [22] to work with our commitments. This is a simple protocol that may provide better concrete efficiency.

1.2 Paper Organization

The remainder of the paper is organized as follows. In Section 2 we present basic notation and computational assumptions used in the paper. In section 3, we define RRC schemes, and in Sections 4 and 5, we present our constructions from LWE and Ring-LWE, respectively. In section 6 we strengthen our security notion and constructions for RRC schemes. In the full version of this paper, we present the generalized BEHG protocol, discuss and construct proofs of shuffle for our RRC schemes, and present additional ways to obtain our stronger security notion. The full version also contains proofs that are omitted from this version.

2 Preliminaries

In this section, we present the basic notions and cryptographic primitives that are used in this work. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. For a distribution X we denote by $x \leftarrow\!\!\$\ X$ the process of sampling a value x from the distribution X. Similarly, for a set $\mathcal{X}$, we denote by $x \leftarrow\!\!\$\ \mathcal{X}$ the process of sampling a value x from the uniform distribution over $\mathcal{X}$. For a pair X, Y of distributions defined over the same domain Ω, we denote by $\mathsf{SD}(X, Y)$ the *statistical distance* between them, defined as $\mathsf{SD}(X, Y) = \frac{1}{2}\sum_{\omega \in \Omega} |\Pr[X = \omega] - \Pr[Y = \omega]|$.

We denote matrices by boldface capital letters, e.g. $\boldsymbol{A}$, and vectors in boldface lower-case letters, e.g. $\boldsymbol{v}$. We may use a non-bold capital letter, e.g. A or V, to describe a matrix or a vector, when we wish to emphasize that this matrix or vector is being treated as a random variable. As standard, we identify $\mathbb{Z}_q$ for a prime q with the set $(-q/2, \ldots, q/2]$, and we define the absolute value of an element $x \in \mathbb{Z}_q$ as $|x| = \{\min |y| : y \in \mathbb{Z}, y = x \pmod{q}\}$.

For $n, p \in \mathbb{N}$ where p is prime, we define the rings $\mathcal{R} = \mathbb{Z}[x]/f(x)$ and $\mathcal{R}_p = \mathcal{R}/\langle p \rangle$, where $f(x)$ is monic and of degree n. That is, $\mathcal{R}_p$ is the ring of polynomials modulo $f(x)$ with integer coefficients in $\mathbb{Z}_p$. We define the norm of elements in these rings to be the norm of their coefficient vector in $\mathbb{Z}^n$, which is also called the coefficient embedding. For any $g(x) = \sum_{i \in 0 \cup [n-1]} \alpha_i x^i \in \mathcal{R}$, we use $\mathsf{coeff}(g)$ to denote the vector $\{\alpha_0, \ldots, \alpha_{n-1}\}$, i.e. the coefficient embedding of $g(x)$, and the norm is defined as follows:

$$||g||_1 = \sum \alpha_i \quad ||g||_2 = (\sum \alpha_i^2)^{1/2} \quad ||g||_\infty = \max|\alpha_i|$$

For a vector $\boldsymbol{v}$ over $\mathcal{R}$, we define $||\boldsymbol{v}|| = (\sum_i ||\boldsymbol{v}_i||^2)^{1/2}$.

2.1 Lattice Assumption

The paper makes use of two basic and standard lattice-based assumptions, the learning with errors (LWE) assumption and the short integer solution (SIS) assumption, both of which over integer lattices. We briefly recall these assumptions here. For a more detailed survey of these assumptions and their hardness, see, for example, [36] and the many references therein.

The LWE assumption. We rely on the following formulation of the learning with errors (LWE) problem, introduced by Regev [39]. The problem is parameterized by a prime modulus q, a vector length n which typically corresponds to the security parameter λ, and a noise distribution χ. For our needs, the important thing is that χ is highly concentrated on low-norm vectors such that with overwhelming probability $\|\boldsymbol{x}\|_2 \leq \delta$ for $\boldsymbol{x} \xleftarrow{\$} \chi$ for some $\delta = \delta(\lambda)$ (one typically takes χ to be a discrete Gaussian with appropriate parameters)

► **Definition 1.** *Let $q = q(\lambda)$ be a prime, $n = n(\lambda)$ be an integer, and $\chi = \chi(\lambda)$ be a distribution over $\mathbb{Z}_q$, all public functions of the security parameter $\lambda \in \mathbb{N}$. The (q, n, χ)-LWE assumption states that for every probabilistic polynomial time algorithm $\mathcal{A}$ and for all polynomially-bounded functions $m = m(\lambda)$ there exists a negligible function $\nu(\cdot)$ such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathrm{lwe}}(\lambda) := |\Pr[\mathcal{A}(\boldsymbol{A}, \boldsymbol{A} \cdot \boldsymbol{s} + \boldsymbol{e}) = 1] - \Pr[\mathcal{A}(\boldsymbol{A}, \boldsymbol{v}) = 1]| \leq \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$, where $\boldsymbol{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times n}, \boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n, e \xleftarrow{\$} \chi^m$, and $\boldsymbol{v} \xleftarrow{\$} \mathbb{Z}_q^m$.

2.2 Ring Lattice Assumption

We will also use the ring-based variant of the LWE assumption, introduced by [33].

The RLWE assumption. This problem is also parameterized by the prime modulus q, degree of the modulus polynomial n, and a noise distribution χ. We focus on a special case of the Ring-LWE problem where $f(x) = x^n + 1$, and n is a power of two. Similar to LWE, χ is highly concentrated on low-norm polynomials such that with overwhelming probability $||x||_2 \leq \delta$ for $x \leftarrow\$ \chi$ for some $\delta = \delta(\lambda)$. χ is usually taken to be a discrete gaussian in the coefficient embedding of $\mathcal{R}$.

► **Definition 2.** *Let $q = q(\lambda)$ be a prime, $n = n(\lambda)$ be an integer, and $\chi = \chi(\lambda)$ be a distribution over $\mathcal{R}$, all public functions of the security parameter $\lambda \in \mathbb{N}$. The (q, n, χ)-RLWE assumption states that for every probabilistic polynomial time algorithm $\mathcal{A}$ and for all polynomially-bounded functions $m = m(\lambda)$, there exists a negligible function $\nu(\cdot)$ such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathrm{rlwe}}(\lambda) := |\Pr[\mathcal{A}(\boldsymbol{a}, \boldsymbol{b}) = 1] - \Pr[\mathcal{A}(\boldsymbol{a}, \boldsymbol{v}) = 1]| \leq \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$, where $\boldsymbol{a} \xleftarrow{\$} \mathcal{R}_q^m, s \xleftarrow{\$} \mathcal{R}_q, \boldsymbol{e} \xleftarrow{\$} \chi^m, \boldsymbol{b}_i = \boldsymbol{a}_i \cdot s + \boldsymbol{e}_i \, \forall \, i \in [m]$, and $\boldsymbol{v} \xleftarrow{\$} \mathcal{R}_q^m$.

2.3 Randomness Extraction

We will use the following lemma from the work of Gentry, Peikert, and Vaikuntanathan [26]. The lemma follows from the leftover hash lemma [29].

▶ **Lemma 3** ([26, 29]). *Let q be a prime and let m, n be integers. Let R, A and B be random variables distributed uniformly in $\{-1,1\}^{m\times m}$, $\mathbb{Z}_q^{m\times n}$, and $\mathbb{Z}_q^{m\times n}$, respectively. Then, it holds that*

$$\mathsf{SD}\left((A, R\cdot A), (A, B)\right) \leq \frac{m}{2}\cdot\sqrt{2^{-m+n\log q}}.$$

When working over polynomial rings, we will not be able to use the leftover hash lemma. Instead, we will use the regularity lemma defined over rings [42].

▶ **Lemma 4** (Generalization of Theorem 3.2, [42]). *Let $\mathbb{F}$ be a finite field and $f \in \mathbb{F}[x]$ be monic and of degree $n > 0$. Let $\mathcal{R}$ be the ring $\mathbb{F}[x]/f$ and $m > 0$. For every $i, j \in [m]$ and $k \in [n]$, let $D_{i,j,k} \subseteq \mathbb{F}$, with $|D_{i,j,k}| = d$. Let A, B be random variables distributed uniformly in $\mathcal{R}^{m\times\ell}$. Let $R \in \mathcal{R}^{m\times m}$ be a matrix of polynomials, wherein the kth coefficient of $R_{i,j}$ is chosen uniformly randomly and independently from $D_{i,j,k}$, for all $i, j \in [m]$ and $k \in [n]$. Then, it holds that,*

$$\mathsf{SD}\left((A, RA), (A, B)\right) \leq \frac{m}{2}\sqrt{\prod_{i\in[t]}\left(1+\left(\frac{|\mathbb{F}|}{d^m}\right)^{\mathsf{deg}(f_i)}\right)^{\ell} - 1}$$

where $f = \prod_{i\in[t]} f_i$ is the factorization of f over $\mathbb{F}[x]$, and $\mathsf{deg}(f_i)$ is the degree of the polynomial f_i.

Specifically, we will choose $\mathbb{F} = \mathbb{Z}_q$ and $D_{i,j,k} = \{-1,1\}\,\forall\, i,j \in [m], k \in [n]$.

We will also rely on the following definition for the norm of a matrix and a related lemma from Agrawal, Boneh, and Boyen [1] (a similar lemma appears in [3]), which states that a random Bernoulli matrix has low norm with overwhelming probability.

▶ **Definition 5.** *Let $\boldsymbol{R}$ be an $m\times m$ matrix over $\mathbb{Z}$. Let $\boldsymbol{B}_m := \{\boldsymbol{x} \in \mathbb{R}^m : \|\boldsymbol{x}\|_2 = 1\}$ be the unit ball in $\mathbb{R}^m$. Define the norm of the matrix $\boldsymbol{R} \in \mathbb{Z}^{m\times m}$ as*

$$\|\boldsymbol{R}\| := \max_{\boldsymbol{x}\in\boldsymbol{B}_m} \|\boldsymbol{R}\cdot x\|_2 .$$

The norm for a matrix in $\mathcal{R}_q^{m\times m}$ is defined similarly. The following two lemmas bound the norm of random matrices where all entries are sampled i.i.d. from a distribution concentrated around 0.

▶ **Lemma 6** ([1, 3]). *Let q be a prime and let m be an integer. Let R be a random variable uniformly sampled from $\{-1,1\}^{m\times m}$. Then, there is a universal constant $C > 0$ such that*

$$\Pr\left[\|R\| \geq C\cdot\sqrt{m}\right] < e^{-2m}.$$

A proof for the following lemma can be found in the full version.

▶ **Lemma 7.** *Let q be a prime and let m, n be integers. Let $R \in \mathcal{R}_q^{m\times m}$ be a random variable, such that for all $i, j \in [m]$, the coefficient vector of $R_{i,j}$ is sampled uniformly at random from $\{-1,1\}^n$. Then,*

$$\Pr\left[\|R\| \geq m\sqrt{mn}\cdot\omega(\sqrt{logn})\right] < negl(n)$$

3 Re-randomizable Commitments

Informally, a re-randomizable commitment (RRC, for short) is a scheme that allows one to commit to random keys.[1] Moreover, an RRC scheme supports re-randomizations of commitments: given a commitment c to a key k, one should be able to re-randomize to commitment to produce a new commitment c' for k. Importantly, knowledge of c suffices for such re-randomization, and no additional secrets are needed. In particular, the re-randomizing entity is not required to know the key k nor the randomness used to create c.

We first present the syntax for RRC schemes and the associated correctness requirement. Then, we discuss two security notions that such schemes should satisfy.

3.1 Syntax & Correctness

An RRC scheme R is a tuple of four algorithms:

- $\mathit{Setup}(1^\lambda) \to pp$: outputs public parameters pp,
- $\mathit{Commit}(pp) \to (c, k)$: outputs a commitment string c and a key k,
- $\mathit{Randomize}(pp, c) \to c'$: randomize the commitment,
- $\mathit{Test}(pp, c, k) \to \{0, 1\}$: outputs 1 if k is a valid key for c.

The first three are probabilistic polynomial time (PPT) and the fourth is deterministic polynomial time.

In terms of correctness, we require that $\mathit{Test}(pp, c, k)$ outputs 1 for (c, k) output by $\mathit{Commit}(pp)$. Moreover, $\mathit{Test}(pp, c', k)$ should output 1 if c' was obtained from c via at most B consecutive re-randomizations, where B is a parameter. We call this correctness requirement *B-randomizability*. If a scheme is B-randomizable for all B, we call it fully-randomizable.

▶ **Definition 8.** *An RRC scheme is* **B-randomizable** *if there exists a negligible function $\nu(\cdot)$ such that the following holds for every $\lambda \in \mathbb{N}$:*
let $pp \xleftarrow{\$} \mathit{Setup}(1^\lambda)$, $(c_0, k) \xleftarrow{\$} \mathit{Commit}(pp)$, and $c_i \xleftarrow{\$} \mathit{Randomize}(pp, c_{i-1})$ for $i = 1, \ldots,$ then

$$\Pr\Big[\mathit{Test}(pp, c_i, k) = 1 \quad \text{for } i = 0, 1, 2, \ldots, B\Big] \geq 1 - \nu(\lambda).$$

An RRC scheme that is B-randomizable for all $B \in \mathbb{N}$ is said to be **fully randomizable**. ⌟

For the notion of RRC schemes to be non-trivial, we require that the key k generated by *Commit* to have high min-entropy.

▶ **Definition 9.** *An RRC scheme is* **B-randomizable** *is* **non-trivial** *if there exists a negligible function $\nu(\cdot)$ such that the following holds for every $\lambda \in \mathbb{N}$: let $pp \xleftarrow{\$} \mathit{Setup}(1^\lambda)$, $(c_0, k_0) \xleftarrow{\$} \mathit{Commit}(pp)$ and $(c_1, k_1) \xleftarrow{\$} \mathit{Commit}(pp)$, then* $\Pr[k_0 = k_1] \leq \nu(\lambda)$. ⌟

3.2 Notions of Security

An RRC scheme should satisfy two security properties: Binding and Unlinkability.

[1] Committing to random keys is sufficient for the main application we consider, which is SSLE protocols. Observe, however, that such a scheme can be easily converted into a scheme that allows one to commit to arbitrary messages via a one-time pad.

```
Game $\mathbf{G}_{\mathcal{A},\mathsf{R}}(\lambda, B)$
1 :  $b \xleftarrow{\$} \{0,1\}$
2 :  $pp \xleftarrow{\$} \mathsf{R}.\mathit{Setup}(1^\lambda)$
3 :  $(c_0, k_0) \xleftarrow{\$} \mathsf{R}.\mathit{Commit}(pp), \quad (c_1, k_1) \xleftarrow{\$} \mathsf{R}.\mathit{Commit}(pp)$
4 :  $(\mathsf{state}, i_0, i_1) \xleftarrow{\$} \mathcal{A}(pp, c_0, c_1)$
5 :  $\textbf{if } (i_0 > B) \text{ OR } (i_1 > B) : \textbf{ abort}$
6 :  $\textbf{if } (i_0 = 0) \text{ OR } (i_1 = 0) : \textbf{ abort}$
7 :  $c \leftarrow c_b$
8 :  $\textbf{for } t \textbf{ in } \{1, \ldots, i_b\} : \quad c \xleftarrow{\$} \mathsf{R}.\mathit{Randomize}(pp, c)$
9 :  $b' \xleftarrow{\$} \mathcal{A}(c, \mathsf{state})$
10 : $\textbf{return } b = b'$
```

Figure 1 The security game for an adversary $\mathcal{A}$ attacking the unlinkability of an RRC scheme R.

Binding. Similarly to standard commitment schemes, we require that a commitment can be tied to at most one key.

▶ **Definition 10.** *An RRC scheme is* **perfectly binding** *if for every* $\lambda \in \mathbb{N}$ *and for all* c, k, k' *we have*

$$\Pr_{pp \xleftarrow{\$} Setup(1^\lambda)}[k \neq k' \quad \text{AND} \quad \mathit{Test}(pp, c, k) = \mathit{Test}(pp, c, k') = 1] = 0. \tag{1}$$

⌟

Condition (1) ensures that a commitment c will *never* be accepted by two distinct keys. As we will later discuss, this is satisfied by the previous DDH-based construction of Boneh et al. [16]. For our lattice-based construction, we need to weaken this condition a bit and only require that (1) holds computationally. This leads to the following definition.

▶ **Definition 11.** *We say that an RRC scheme is* **computationally binding** *if for all* PPT *adversaries* $\mathcal{A}$ *the following function is negligible.*

$$\Pr\left[k \neq k' \quad \text{AND} \quad \mathit{Test}(pp, c, k) = \mathit{Test}(pp, c, k') = 1 \ : \ \begin{array}{l} pp \xleftarrow{\$} \mathit{Setup}(1^\lambda) \\ (c, k, k') \xleftarrow{\$} \mathcal{A}(pp) \end{array}\right] \tag{2}$$

⌟

Unlinkability. An RRC scheme R is unlinkable if a PPT adversary is unable to distinguish the i-th re-randomization of a commitment c_0 from the j-th re-randomization of another commitment c_1. This is captured in the security game in Figure 1. As usual, we define the adversary's advantage as

$$\mathsf{Adv}^{\mathrm{rrc}}_{\mathcal{A},\mathsf{R},B}(\lambda) := \left|2\Pr[\mathbf{G}_{\mathcal{A},\mathsf{R}}(\lambda, B) = 1] - 1\right|.$$

▶ **Definition 12.** *A* B*-randomizable RRC scheme is* **unlinkable** *if for all* PPT *adversaries* $\mathcal{A}$ *the function* $\mathsf{Adv}^{\mathrm{rrc}}_{\mathcal{A},\mathsf{R},B}(\lambda)$ *is negligible.*

We make two remarks on the unlinkability definition:

- Looking ahead, for some applications, we might want the scheme to remain unlinkable even if adversarial re-randomizations were applied to it at some point. We present such a definition in Section 6. We also discuss ways to augment our basic LWE-based and Ring-LWE-based constructions to accommodate this stronger security definition. Since the stronger unlinkability definition is much more complicated than the one in Fig. 1, we first focus on this weaker notion.
- An unlinkable RRC scheme is, in particular, *hiding*. Meaning, that a commitment c leaks no information (in a computational sense) regarding the committed key k. Intuitively, an adversary that can distinguish between a commitment to a key k and a commitment to a different key k' can trivially link a commitment c to either a commitment c_0 to k_0 or to a commitment c_1 to k_1 by outputting the bit b such that c is a commitment to k_b.

3.3 An RRC scheme based on DDH

Equipped with the above definitions, we can briefly recall the DDH-based RRC scheme used in [16]. The scheme, called $\mathsf{R}_{\mathrm{ddh}}$, is defined by:

- $\mathit{Setup}(1^\lambda)$: choose a finite cyclic group $\mathbb{G}$ with generator $g \in \mathbb{G}$ and output $pp := (\mathbb{G}, g)$.
- $\mathit{Commit}(pp)$: choose random $u \xleftarrow{\$} \mathbb{G}$ and $k \xleftarrow{\$} \mathbb{Z}_q$, set $c \leftarrow (u, u^k)$, and output (c, k).
- $\mathit{Randomize}(pp, c)$: parse $c = (u, v)$, choose a random $\rho \xleftarrow{\$} \mathbb{Z}_q$, and output $c' := (u^\rho, v^\rho)$.
- $\mathit{Test}(pp, c, k)$: parse $c = (u, v)$ and output 1 iff $u^k = v$, otherwise output 0.

▶ **Theorem 13** ([16]). *If the DDH assumption holds in $\mathbb{G}$ then R_{ddh} is a perfectly-binding, unlinkable, and fully randomizable RRC.*

The fact that the scheme is full-randomizable and perfectly binding is easy to observe. The proof of unlinkability is a direct application of DDH. In the next section, we construct an RRC scheme that is post-quantum secure based on the LWE assumption.

4 A Construction from Learning with Errors

In this section, we present a construction of an RRC scheme from the LWE assumption [39] (see Section 2). An informal overview of the construction is presented in Section 1.1.

4.1 The Construction

Our construction of an RRC scheme from LWE, denoted $\mathsf{R}_{\mathsf{lwe}}$ is presented in Fig. 2. The construction is parameterized by an integer B, which serves as a bound on the number of re-randomizations that can be applied to a commitment. In the construction, we use Δ to denote $(C \cdot \sqrt{m})^B \cdot \delta$, where m is a parameter of the scheme determined by the analysis (think of $m = O(\lambda)$), C is the universal constant from Lemma 6 and δ is a bound on the ℓ_2 norm of the LWE noise vectors used in the construction.

Correctness. First, note that prior to any randomization being preformed, for an honestly-generated commitment $c = (\boldsymbol{A}, \boldsymbol{U})$ it holds that $\boldsymbol{A} \cdot \mathsf{H}(k) - \boldsymbol{U}$ is equal to the noise matrix $\boldsymbol{E}$ sampled according to $\chi^{m\times\ell}$ during the generation of the commitment. Hence, the matrix computed by the *Test* algorithm is simply $\boldsymbol{E}$, and each of its columns has norm at most δ. Now, after $t \le B$ applications of *Randomize* to c using matrices $\boldsymbol{R}_1, \dots, \boldsymbol{R}_t$, the commitment we get is of the form

$$(\boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{A}, \boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{U}) = (\boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{A}, \boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{A} \cdot \mathsf{H}(k) + \boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{E}).$$

```
- $Setup(1^\lambda)$:
    1 : Let $n := \lambda$, choose a prime $q$, and choose $m = m(n,q)$ and $\ell = \ell(n,q)$.
          // we will explain how to choose $m$ and $\ell$ in the analysis
    2 : Let $\chi$ be the LWE noise distribution over $\mathbb{Z}_q$.
          // if $\boldsymbol{e} \xleftarrow{\$} \chi^m$, and we lift $\boldsymbol{e}$ to $\mathbb{Z}^m$, then with high probability, $\|\boldsymbol{e}\|_2 \le \delta$ for some $\delta \ll q$
    3 : return $pp \leftarrow (\lambda, q, n, m, \ell, \chi)$
- $Commit(pp)$:
    1 : $\boldsymbol{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$   // choose a random matrix $\boldsymbol{A}$
    2 : $k \xleftarrow{\$} \{0,1\}^{1^\lambda}$   // choose a random $\lambda$-bit string
    3 : $\boldsymbol{V} \leftarrow \mathsf{H}(k) \in \mathbb{Z}_q^{n \times \ell}$   // hash $k$ to an $n$-by-$\ell$ matrix
    4 : sample $\boldsymbol{E} \in \mathbb{Z}_q^{m \times \ell}$ from the LWE noise distribution $\chi^{m \times \ell}$
          // then for each column $\boldsymbol{e}$ of $\boldsymbol{E}$, $\|\boldsymbol{e}\|_2 \le \delta$ w.h.p when $\boldsymbol{e}$ is lifted to $\mathbb{Z}^m$
    5 : $\boldsymbol{U} \leftarrow \boldsymbol{A} \cdot \boldsymbol{V} + \boldsymbol{E} \in \mathbb{Z}_q^{m \times \ell}$
    6 : $c \leftarrow (\boldsymbol{A}, \boldsymbol{U})$
    7 : return $(c, k)$
- $Randomize(pp, c)$: parse $c = (\boldsymbol{A}, \boldsymbol{U})$ and do
    1 : sample a random matrix $\boldsymbol{R} \xleftarrow{\$} \{-1,1\}^{m \times m}$   // $\boldsymbol{R}$ is a low-norm matrix
    2 : $c' \leftarrow (\boldsymbol{R} \cdot \boldsymbol{A},\ \boldsymbol{R} \cdot \boldsymbol{U}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times \ell}$
    3 : return $c'$
- $Test(pp, c, k)$: parse $c = (\boldsymbol{A}, \boldsymbol{U})$ and do
    1 : $\boldsymbol{V} \leftarrow \boldsymbol{A} \cdot \mathsf{H}(k) - \boldsymbol{U} \in \mathbb{Z}_q^{m \times \ell}$
    2 : return 1 iff for each column $\boldsymbol{v}$ of $\boldsymbol{V}$, $\|\boldsymbol{v}\|_2 \le \Delta$ when $\boldsymbol{v}$ is lifted to $\mathbb{Z}^m$
        Otherwise, return 0
```

Figure 2 $\mathsf{R}_{\mathsf{lwe}}$ – A B-randomizable RRC scheme based on the learning with errors (LWE) problem.

Hence, the matrix computed by the $Test$ algorithm is $\boldsymbol{E}' = \boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{E}$. Since $\boldsymbol{R}_1, \ldots, \boldsymbol{R}_t$ are sampled independently from $\{-1,1\}^{m \times m}$, Lemma 6 guarantees that with overwhelming probability, each column of $\boldsymbol{E}'$ has norm at most $(C \cdot \sqrt{m})^t \cdot \delta \le (C \cdot \sqrt{m})^B \cdot \delta = \Delta$.

4.2 Binding

▶ **Theorem 14.** *The above scheme is computationally binding when* H *is modeled as a random oracle. Concretely, for every adversary* $\mathcal{A}$ *making at most* Q *queries to* H *it holds that*

$$\Pr\left[\begin{array}{c} k \neq k' \text{ AND} \\ Test(pp,c,k) = Test(pp,c,k') = 1 \end{array} : \begin{array}{c} pp \xleftarrow{\$} Setup(1^\lambda) \\ (c,k,k') \xleftarrow{\$} \mathcal{A}(pp) \end{array}\right] \le Q^2 \cdot q^n \cdot \left(\frac{4\Delta + 1}{q}\right)^\ell$$

The proof of Theorem 14 can be found in the full version.

4.3 Unlinkability

▶ **Theorem 15.** *The above construction is unlinkable. In particular, for every* PPT *adversary* $\mathcal{A}$ *making at most* $Q = Q(\lambda)$ *queries to* H*, there exists a* PPT *adversary* $\mathcal{B}$ *such that for all* $\lambda \in \mathbb{N}$ *it holds that:*

$$\mathsf{Adv}^{\mathrm{rrc}}_{\mathcal{A},\mathsf{R}_{\mathsf{LWE}},B}(\lambda) \leq \frac{2Q}{2^{\lambda}-Q} + 2\ell \cdot \mathsf{Adv}^{\mathrm{lwe}}_{\mathcal{B}}(\lambda) + \frac{B \cdot m}{2} \cdot \sqrt{2^{-m+(n+\ell)\cdot \log q}}.$$

The proof of the theorem is in the full version.

5 A Construction from Ring LWE

In this section we present our RRC construction from the Ring LWE assumption [33] (see Section 2). Our construction, denoted $\mathsf{R}_{\mathsf{rlwe}}$ is presented in Fig. 3. The construction works in a polynomial ring $\mathcal{R}$ modulo a cyclotomic polynomial f that has exactly two irreducible factors f_1, f_2 over $\mathbb{Z}_q$.

Improvements over $\mathsf{R}_{\mathsf{LWE}}$. Compared to the integer-based scheme, the ring-based scheme accommodates more efficient parameter choices. For concreteness, the ensuing discussion focuses on the regime in which $q = \Omega(\Delta^2)$. In this regime, for the ring-based scheme to be binding, we only need ℓ to be $\Omega(\log(q) + \lambda/n)$, where λ is the security parameter. This is a factor of $\Omega(n)$ smaller than the LWE case. Secondly, m only needs to be of order $\Omega(\log(q) + (\ell + \kappa)/n)$ where κ is a statistical security parameter (we want the re-randomized commitments to be distributed $1/2^{\kappa}$ close to a uniform distribution). This also turns out to be a factor of $\Omega(n)$ smaller than the integer case. Combining these together, each ring-based RRC commitment and each re-randomization matrix is $\Omega(n)$-times smaller than the integer-based commitment and matrix, respectively (this already takes into account the fact that representing each ring element takes n-times the representation length of a $\mathbb{Z}_q$ element).

Additionally, if we are re-randomizing a list of t commitments, then we consider the possibility of using a single, larger matrix to re-randomize all the commitments. In the ring case, we would only need to scale m by a factor of t/n, but in the integer case, m grows by a factor of t. In particular, for $t = \Omega(n)$, the ring-based RRC commitments and the re-randomization matrix only grow by a constant factor, while in the integer case, the commitments and the re-randomization matrix still grow linearly in t (making it essentially infeasible to use a single re-randomization matrix in this setting)[2].

We now prove the correctness, binding, and unlinkability for our ring-based RRC scheme.

Correctness. We first note that, prior to any rerandomization, for an honestly generated commitment $c = (\boldsymbol{a}, \boldsymbol{U})$, it holds that $\boldsymbol{U} - \boldsymbol{a} \cdot \mathsf{H}(k)^T$ is equal to the noise matrix $\boldsymbol{E}$ sampled at the generation of the commitments. Hence, the matrix $\boldsymbol{V}$ computed by the *Test* algorithm is just $\boldsymbol{E}$, and each of its columns has norm at most δ, since $\boldsymbol{E}$ was sampled according to $\chi^{m \times \ell}$. Now, after $t \leq B$ applications of *Randomize* to the commitment c using matrices $\boldsymbol{R}_1, \ldots, \boldsymbol{R}_t$, the commitment we get is of the form

$$(\boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{a}, \boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{U}) = (\boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{a}, \boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{a} \cdot \mathsf{H}(k)^T + \boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{E}).$$

Hence, the matrix computed by the *Test* algorithm is $\boldsymbol{E}' = \boldsymbol{R}_t \cdots \boldsymbol{R}_1 \cdot \boldsymbol{E}$. Lemma 7 guarantees that with a high probability, each column of $\boldsymbol{E}'$ has norm at most

$$(m\sqrt{mn} \cdot \omega(\sqrt{\log n}))^t \cdot \delta \leq (m\sqrt{mn} \cdot \omega(\sqrt{\log n}))^B \cdot \delta = \Delta$$

where Δ is an upper bound on the expression $\delta \cdot (m\sqrt{mn} \cdot \omega(\sqrt{\log n}))^B$.

[2] While the re-randomization matrix would typically not be transmitted in the clear, its size does affect the complexity of the proof of shuffle (recall our discussion in the introduction).

- *Setup*(1^λ):
```
1 : Let $n := 2^r$ where $r = r(\lambda)$, and let $f(x) = x^n + 1$ and $\mathcal{R} = \mathbb{Z}[x]/f(x)$.
    Choose a prime $q$ and let $\mathcal{R}_q = \mathbb{Z}_q[x]/f(x)$.
      // $r, q$ are chosen such that $f$ factors into two irreducible polynomials over $\mathbb{Z}_q$
2 : Let $m = m(n, q)$ and $\ell = \ell(n, q)$.
      // we will explain how to choose $m$ and $\ell$ in the analysis
3 : Let $\chi$ be the RLWE noise distribution over $\mathcal{R}_q$.
      // if we sample a vector $\boldsymbol{e} \xleftarrow{\$} \chi^m$, then with high probability, $\|\boldsymbol{e}\|_2 \leq \delta$ for some $\delta \ll q$
4 : return $pp \leftarrow (\lambda, q, n, m, \ell, \chi)$
```
- *Commit*(pp):
```
1 : $\boldsymbol{a} \xleftarrow{\$} \mathcal{R}_q^m$   // choose a random vector $\boldsymbol{a}$
2 : $k \xleftarrow{\$} \{0,1\}^\lambda$   // choose a random $1^\lambda$-bit string
3 : $\boldsymbol{v} \leftarrow \mathsf{H}(k) \in \mathcal{R}_q^\ell$   // hash $k$ to a vector of length $\ell$
4 : Sample $\boldsymbol{E} \in \mathcal{R}_q^{m\times\ell}$ from the RLWE noise distribution $\chi^{m\times\ell}$
      // then for each column $\boldsymbol{e}$ of $\boldsymbol{E}$, $\|\boldsymbol{e}\|_2 \leq \delta$ w.h.p
5 : $\boldsymbol{U} \leftarrow \boldsymbol{a} \cdot \boldsymbol{v}^T + \boldsymbol{E} \in \mathcal{R}_q^{m\times\ell}$
6 : $c \leftarrow (\boldsymbol{a}, \boldsymbol{U})$
7 : return $(c, k)$
```
- *Randomize*(pp, c): parse $c = (\boldsymbol{a}, \boldsymbol{U})$ and do
```
1 : Sample $\boldsymbol{R} \in \mathcal{R}^{m\times m}$:
    $\forall i, j \in [m]$, sample the coefficients of $\boldsymbol{R}_{i,j}$ uniformly and independently from $\{-1, 1\}$
        // $\boldsymbol{R}$ is a low-norm matrix
2 : $c' \leftarrow (\boldsymbol{R}\cdot\boldsymbol{a},\ \boldsymbol{R}\cdot\boldsymbol{U}) \in \mathcal{R}_q^m \times \mathcal{R}_q^{m\times\ell}$
3 : return $c'$
```
- *Test*(pp, c, k): parse $c = (\boldsymbol{a}, \boldsymbol{U})$ and do
```
1 : $\boldsymbol{V} \leftarrow \boldsymbol{U} - \boldsymbol{a}\cdot\mathsf{H}(k)^T \in \mathcal{R}_q^{m\times\ell}$
2 : return 1 iff for each column $\boldsymbol{v}$ of $\boldsymbol{V}$, $\|\boldsymbol{v}\|_2 \leq \Delta$, and return 0 otherwise
```

Figure 3 $\mathsf{R}_{\mathsf{rlwe}}$ – A B-randomizable RRC scheme based on the learning with errors over rings (RLWE) problem.

5.1 Binding

▶ **Theorem 16.** *The above scheme is computationally binding when* H *is modeled as a random oracle. Concretely, for every adversary $\mathcal{A}$ making at most Q queries to* H *it holds that*

$$\Pr\left[\begin{array}{c} k \neq k' \text{ AND} \\ Test(pp, c, k) = Test(pp, c, k') = 1 \end{array} : \begin{array}{c} pp \xleftarrow{\$} Setup(1^\lambda) \\ (c, k, k') \xleftarrow{\$} \mathcal{A}(pp) \end{array}\right] \leq Q^2 \cdot q^n \cdot \left(\frac{4\Delta + 1}{\sqrt{q}}\right)^{n\ell}$$

The proof of Theorem 16 is in the full version.

5.2 Unlinkability

▶ **Theorem 17.** *The above construction is unlinkable. In particular, for every* PPT *adversary $\mathcal{A}$ making at most $Q = Q(\lambda)$ queries to* H*, there exists a* PPT *adversary $\mathcal{B}$ such that for all $\lambda \in \mathbb{N}$ it holds that:*

$$\mathsf{Adv}^{\mathrm{rrc}}_{\mathcal{A},\mathsf{R}_{\mathsf{RLWE}},B}(\lambda) \leq \frac{2Q}{2^\lambda - Q} + (2\ell) \cdot \mathsf{Adv}^{\mathrm{rlwe}}_{\mathcal{B}}(\lambda) + B \cdot \frac{m}{2}\sqrt{\left(1 + \left(\frac{q}{2^m}\right)^{n/2}\right)^{2(\ell+1)} - 1}.$$

The proof of Theorem 17 can be found in the full version.

6 Handling Adversarially-Randomized Commitments

In this section, we present a stronger notion of unlinkability, called *strong unlinkability* for RRC schemes, and then present different approaches to augment our basic schemes from Sections 4 and 5 to satisfy this definition.

Loosely speaking, strong unlinkability requires that re-randomization should result in unlinkable commitments, even if they were previously re-randomized by the adversary. This trivially holds for the DDH-based construction of Boneh et al. [16] thanks to two properties of the scheme:

- Suppose the adversary receives a commitment c for which k is a valid key, and outputs a randomized commitment c'. As long as $Test(pp, c', k) = 1$, there exists some randomness r such that $c' = Randomize(pp, c; r)$.
- Re-randomization using *Randomize* is a commutative operation. Hence, in conjunction with the observation above, any knowledge the adversary could gain by re-randomizing a commitment before an honest re-randomization, it could also gain by re-randomizing it afterwards (which the adversary can already do in the security game from Fig. 1).

Alas, this is not the case for our lattice-based constructions. The main issue is that matrix multiplication is not commutative. Hence a "bad" re-randomization (even one that does not invalidate the honest commitment key) can have a long lasting effect on a commitment even after many subsequent honest re-randomizations have taken place. Concretely, on input $c = (\boldsymbol{A}, \boldsymbol{U})$, the adversary may output $c' = (\boldsymbol{A}', \boldsymbol{U}')$, such that $\boldsymbol{A}'$ is "bad" in the sense that the distribution $R \cdot \boldsymbol{A}'$ for a random $R \xleftarrow{\$} \{-1, 1, \}^{m \times m}$ is very far from the uniform distribution over $\mathbb{Z}_q^{m \times n}$. As a hypothetical example, suppose that the adversary can find a matrix $\boldsymbol{R} \in \{-1, 1, \}^{m \times m}$ such that $\boldsymbol{A}' = \boldsymbol{R} \cdot \boldsymbol{A}$ is a low-norm matrix. Then, the distribution $R \cdot \boldsymbol{A}'$ will be concentrated on low-norm matrices as well, enabling the adversary to distinguish between this distribution and the uniform distribution over $\mathbb{Z}_q^{m \times n}$, which is concentrated on high-norm matrices.

6.1 A Stronger Unlinkability Definition

We first need to define what it means for an RRC scheme to be unlinkable in the face of adversarial re-randomizations. To do this, we augment the security game of RRC schemes by letting the adversary re-randomize the commitments at points in time of its choosing. To avoid trivial attacks, we require that the adversary justifies its outputs by providing the randomness it used for re-randomization.

To this end, and to facilitate our constructions, we introduce several new notions for RRC schemes:

- We augment an RRC scheme with a corresponding *beacon distribution* D. This distribution is used to model a randomness beacon, and will be used by one of our constructions of a strongly-unlinkable RRC scheme. In practice, the beacon may be assumed as an outside resource or implemented in various ways using known techniques [37].
- We introduce two new algorithms $\mathsf{R}.\mathit{Precommit}$ and $\mathsf{R}.\mathit{Extract}$ to an RRC scheme R. $\mathsf{R}.\mathit{Precommit}$ is a randomized algorithm that takes in the public parameters pp an outputs some "precommitment" pcom, whose role will become apparent in a minute. $\mathsf{R}.\mathit{Extract}$

is a (potentially randomized) algorithm that takes in pp, the randomness $r \in \{0,1\}^*$ used by R.*Precommit* to generate pcom, and a sample rand from D, and outputs some randomness r' to be used by R.*Randomize*. Throughout this section, we will denote the number of random coins used by R.*Precommit* by $\rho = \rho(\lambda)$.

- An RRC scheme R is now also parameterized by a class $\mathcal{G}$ of *admissible random strings* , and only members of $\mathcal{G}$ can be used as randomness for R.*Randomize*. This is checked by the security game for randomness used by the adversary. A natural selection for $\mathcal{G}$ is the entire support of the randomness used by the honest *Randomize* algorithm; for example, in our (integer) LWE-based construction, this corresponds to $\mathcal{G} = \{-1, 1\}^{m\times m}$, but one might also consider strict supersets or subsets of this set.
  We allow $\mathcal{G}$ to depend on a precommitment pcom, the randomness $r \in \{0,1\}^*$ used by R.*Precommit* to generate pcom, and a sample rand from D. We denote this by $\mathcal{A}(\mathsf{pcom}, r, \mathsf{rand})$. The set $\mathcal{G}$ may also depend on the public parameters pp, but we do not note this explicitly, since the public parameters typically remain fixed.

To recap, an RRC scheme R now consists of six algorithms (R.*Setup*, R.*Commit*, R.*Randomize*, R.*Test*, and now also R.*Precommit* and R.*Extract*), a distribution D, and a set $\mathcal{G} = \mathcal{G}(\mathsf{pcom}, r, \mathsf{rand})$.

Correctness and unlinkability. For correctness, we now require that the scheme is B-rerandomizable (Definition 8), where the randomness for rerandomization is generated by *Precommit*, D, and *Extract*. We additionally require that honestly generated randomness for *Randomize* is indeed admissible.

▶ **Definition 18.** *Let* R *be an RRC scheme such that* R.*Precommit takes* $\rho = \rho(\lambda)$ *random coins. We say* R*is* B**-randomizable** *if there exists a negligible function* $\nu(\cdot)$ *such that the following conditions hold for every* $\lambda \in \mathbb{N}$*:*

1. *Let* $pp \xleftarrow{\$} \mathsf{R}.Setup(1^\lambda)$, $(c_0, k) \xleftarrow{\$} \mathsf{R}.Commit(pp)$, $r_i \xleftarrow{\$} \{0,1\}^\rho$, $\mathsf{rand}_i \xleftarrow{\$} D$, $r'_i \xleftarrow{\$} \mathsf{R}.Extract(pp, r_i, \mathsf{rand}_i)$, $c_i \xleftarrow{\$} \mathsf{R}.Randomize(pp, c_{i-1}; r'_i)$ *for* $i \in [B]$*, then*
$$\Pr\Big[\mathsf{R}.Test(pp, c_i, k) = 1 \quad \text{for } i = 0, 1, 2, \ldots, B\Big] \geq 1 - \nu(\lambda).$$
2. *Let* $pp \xleftarrow{\$} \mathsf{R}.Setup(1^\lambda)$, $r \xleftarrow{\$} \{0,1\}^\rho$, $\mathsf{pcom} \leftarrow \mathsf{R}.Precommit(pp; r)$, $\mathsf{rand} \xleftarrow{\$} D$*, and* $r' \xleftarrow{\$} \mathsf{R}.Extract(pp, r, \mathsf{rand})$*, then*
$$\Pr\left[r' \in \mathcal{G}(\mathsf{pcom}, r, \mathsf{rand})\right] \geq 1 - \nu(\lambda).$$

An RRC scheme that is B*-randomizable for all* $B \in \mathbb{N}$ *is said to be* **fully randomizable**. ⌟

The new strong-unlinkability game is defined in Figure 4. It uses the following abbreviated writing: we write $(\mathsf{rand}, c') \xleftarrow{\$} \mathsf{R}.Randomize(pp, r, c)$ as a shorthand for the process of (1) sampling $\mathsf{rand} \xleftarrow{\$} D$, (3) sampling $r' \xleftarrow{\$} \mathsf{R}.Extract(pp, r, \mathsf{rand})$, (4) computing $c' \leftarrow \mathsf{R}.Randomize(pp, c; r')$, and (5) outputting (rand, c'). The new game is obtained from the old unlinkability security game (Figure 1) by the following modifications:

1. At the onset of the game, the challenger samples precommitments {pcom} to be used for the honest re-randomizations it performs. The adversary then also outputs a precommitment {pcom} for its own future re-randomizations. For each re-randomization, the set $\mathcal{G}$ will depend on the corresponding precommitment. Looking ahead, in a couple of our constructions, the precommitments will serve as commitments for randomness to be used in the future re-randomizations.

2. The challenger samples $\{\mathsf{rand}\}$ values from the beacon distribution D. These serve as the beacon values for each re-randomization (adversarial or honest). The adversary receives the corresponding rand value before each adversarial re-randomization, and together with each honest re-randomization.
3. Each time the adversary $\mathcal{A}$ outputs re-randomized commitments, it also outputs the associated randomness used to generate the associated precommitment and the randomness used for re-randomization. The challenger then checks that this randomness is indeed admissible.

As before, we define the adversary's advantage as

$$\mathsf{Adv}^{\text{strong-rrc}}_{\mathcal{A},\mathsf{R}}(\lambda) := \left|2\Pr[\mathbf{G}^{\text{strong}}_{\mathcal{A},\mathsf{R}}(\lambda) = 1] - 1\right|.$$

```
Game $\boldsymbol{G}^{\text{strong}}_{\mathcal{A},\mathsf{R}}(\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$pp \xleftarrow{\$} \mathsf{R}.\mathit{Setup}(1^\lambda)$
$(T, i_0^{(1)}, i_1^{(1)} \dots, i_0^{(T)}, i_1^{(T)}, \mathsf{state}) \xleftarrow{\$} \mathcal{A}(pp)$
$\overrightarrow{\mathsf{pcom}} \leftarrow ()$   // initialize an empty vector
for $t$ in $\{1,\dots,T\}$ :
    for $j$ in $\{1,\dots,i_0^{(t)}\}$ :  $r_{t,0,j} \xleftarrow{\$} \{0,1\}^\rho$,  $\mathsf{pcom}_{t,0,j} \leftarrow \mathsf{R}.\mathit{Precommit}(pp; r_{t,0,j})$,  $\overrightarrow{\mathsf{pcom}} \leftarrow \overrightarrow{\mathsf{pcom}} \| \mathsf{pcom}_{t,0,j}$
    for $j$ in $\{1,\dots,i_1^{(t)}\}$ :  $r_{t,1,j} \xleftarrow{\$} \{0,1\}^\rho$,  $\mathsf{pcom}_{t,1,j} \leftarrow \mathsf{R}.\mathit{Precommit}(pp; r_{t,1,j})$,  $\overrightarrow{\mathsf{pcom}} \leftarrow \overrightarrow{\mathsf{pcom}} \| \mathsf{pcom}_{t,1,j}$
        // $\rho$ denotes the number of random coins used by $\mathsf{R}.\mathit{Precommit}$
$(\mathsf{pcom}'_{1,0}, \mathsf{pcom}'_{1,1}, \dots, \mathsf{pcom}'_{T,0}, \mathsf{pcom}'_{T,1}, \mathsf{state}) \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \overrightarrow{\mathsf{pcom}})$
$\mathsf{rand}_{1,0}, \mathsf{rand}_{1,1} \dots, \mathsf{rand}_{T,0}, \mathsf{rand}_{T,1} \xleftarrow{\$} D$
$(c_0^{(0)}, k_0) \xleftarrow{\$} \mathsf{R}.\mathit{Commit}(pp)$,   $(c_1^{(0)}, k_1) \xleftarrow{\$} \mathsf{R}.\mathit{Commit}(pp)$
$\mathsf{aux}_0 \leftarrow c_0^{(0)} \| c_1^{(0)}$
for $t$ in $\{1,\dots,T\}$ :
    $(\mathsf{state}, c_0^{(t)}, c_1^{(t)}, r_0, r_1, r'_0, r'_1) \xleftarrow{\$} \mathcal{A}(\mathsf{state}, \mathsf{aux}_{t-1}, \mathsf{rand}_{t,1}, \mathsf{rand}_{t,0})$
        // $r_0$ and $r_1$ are the random coins $\mathcal{A}$ claims to have used to generate $\mathsf{pcom}_{t,0}$ and $\mathsf{pcom}_{t,1}$
        // $r'_0$ and $r'_1$ are the random coins $\mathcal{A}$ claims to have used for re-randomization
    if $(c_0^{(t)} \neq \mathsf{R}.\mathit{Randomize}(pp, c_0^{(t-1)}; r'_0))$ OR $(c_1^{(t)} \neq \mathsf{R}.\mathit{Randomize}(pp, c_1^{(t-1)}; r'_1))$ :  abort
        // check that $r'_0$ and $r'_1$ were used by $\mathcal{A}$ for re-randomization
    for $d$ in $\{0,1\}$  :  $\mathcal{G}_d \leftarrow \mathcal{G}(\mathsf{pcom}'_{t,d}, r_d, \mathsf{rand}_{t,d})$
    $\mathsf{aux}_t \leftarrow ()$   // initialize an empty vector
    for $j$ in $\{1,\dots,i_0^{(t)}\}$ :  $(\mathsf{rand}^t_{0,j}, c_0^{(t)}) \xleftarrow{\$} \mathsf{R}.\mathit{Randomize}(pp, r_{t,0,j}, c_0^{(t)})$,  $\mathsf{aux}_t \leftarrow \mathsf{aux}_t \| (\mathsf{rand}^t_{0,j}, c_0^{(t)})$
    for $j$ in $\{1,\dots,i_1^{(t)}\}$ :  $(\mathsf{rand}^t_{1,j}, c_1^{(t)}) \xleftarrow{\$} \mathsf{R}.\mathit{Randomize}(pp, r_{t,1,j}, c_1^{(t)})$,  $\mathsf{aux}_t \leftarrow \mathsf{aux}_t \| (\mathsf{rand}^t_{1,j}, c_1^{(t)})$
        // the notation $(\mathsf{rand}, c') \xleftarrow{\$} \mathsf{R}.\mathit{Randomize}(pp, r, c)$ is defined above
    if $r'_0 \notin \mathcal{G}_0$ OR $r'_1 \notin \mathcal{G}_1$ :  $c_0^{(t)} \leftarrow c_0^{(t-1)}$,  $c_1^{(t)} \leftarrow c_1^{(t-1)}$
$(\mathsf{rand}_0, c_0) \xleftarrow{\$} \mathsf{R}.\mathit{Randomize}(pp, c_0^{(T)})$,  $(\mathsf{rand}_1, c_1) \xleftarrow{\$} \mathsf{R}.\mathit{Randomize}(pp, c_1^{(T)})$
$b' \xleftarrow{\$} \mathcal{A}(c_b, c_{1-b}, \mathsf{rand}_0, \mathsf{rand}_1, \mathsf{state})$
return $b = b'$
```

Figure 4 The strong unlinkability security game for an adversary $\mathcal{A}$ and an RRC scheme R.

▶ **Definition 19.** *An RRC scheme* R *is* **strongly-unlinkable** *if for all* PPT *adversaries* $\mathcal{A}$ *the function* $\mathsf{Adv}^{\text{strong-rrc}}_{\mathcal{A},\mathsf{R}}(\lambda)$ *is negligible.*

```
Game $\mathbf{G}^{\mathrm{pr}}_{\mathcal{A},\mathsf{R}}(\lambda)$
1 :  $b \xleftarrow{\$} \{0,1\}$
2 :  $pp \xleftarrow{\$} \mathsf{R}.\mathit{Setup}(1^\lambda)$
3 :  $r \xleftarrow{\$} \{0,1\}^\rho$,  $\mathsf{pcom} \leftarrow \mathsf{R}.\mathit{Precommit}(pp; r)$
4 :  $(\mathsf{pcom}', \mathsf{state}) \xleftarrow{\$} \mathcal{A}(pp, \mathsf{pcom})$
5 :  $\mathsf{rand} \xleftarrow{\$} D$
6 :  $(c, k) \xleftarrow{\$} \mathsf{R}.\mathit{Commit}(pp)$,  $(c_0', k') \xleftarrow{\$} \mathsf{R}.\mathit{Commit}(pp)$
7 :  $(c'', r', r'', \mathsf{state}) \xleftarrow{\$} \mathcal{A}(\mathsf{state}, c, \mathsf{rand})$
8 :  if $c'' \neq \mathsf{R}.\mathit{Randomize}(pp, c; r'')$ :  abort
9 :  if $r'' \notin \mathcal{G}(\mathsf{pcom}', r', \mathsf{rand})$ :  abort
10 : $(\mathsf{rand}', c_0) \xleftarrow{\$} \mathsf{R}.\mathit{Randomize}(pp, r, c'')$
11 : $c_1, c_1' \xleftarrow{\$} \mathcal{C}_\lambda$
12 : $b' \xleftarrow{\$} \mathcal{A}(c_b, c_b', \mathsf{rand}', \mathsf{state})$
13 : return $b = b'$
```

Figure 5 The security game for an adversary $\mathcal{A}$ attacking the strong pseudorandomness of R.

How to put the strong unlinkability definition to use. In the strengthened security game from Fig. 4, whenever the adversary re-randomizes, it also sends to the challenger the randomness that went into this re-randomization process (that is, the randomness that went into *Precommit* and into *Randomize*). This means that whenever using a strongly-unlinkable RRC scheme within a larger protocol, one should require that re-randomizers provide a argument of knowledge for such randomness (and potentially of additional secrets that are related to the larger super-protocol). Then, a security reduction that tries to break the security of the RRC scheme can use the knowledge extractor of the proof system to extract the randomness and output it in the RRC security game. Our SSLE protocol, detailed in the full version, provides an example of how to use RRC schemes within a larger protocol. In the full version, we discuss specific ways to construct the necessary arguments of knowledge for our lattice-based RRC schemes.

Strongly-pseudorandom RRC schemes. We present the notion of strong pseudorandomness for RRC schemes. Roughly speaking, an RRC scheme enjoys strong pseudorandomness, if honestly re-randomized commitments are pseudorandom. That is, it is indistinguishable from a uniformly-random member of the domain $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ of commitments. Moreover, honest re-randomization should output pseudorandom commitments even on commitments that were previously re-randomized by the adversary (using admissible randomness). This is captured by the security game in Fig. 5.

As before, we define the adversary's advantage as

$$\mathsf{Adv}^{\text{pr-rrc}}_{\mathcal{A},\mathsf{R}}(\lambda) := \left|2\Pr[\mathbf{G}^{\mathrm{pr}}_{\mathcal{A},\mathsf{R}}(\lambda) = 1] - 1\right|.$$

▶ **Definition 20.** *A B-randomizable* R *scheme is* **strongly-pseudorandom** *if for all* PPT *adversaries $\mathcal{A}$ the function* $\mathsf{Adv}^{\text{pr-rrc}}_{\mathcal{A},\mathsf{R}}(\lambda)$ *is negligible.*

A simple hybrid argument shows that an RRC scheme that is strongly-pseudorandom is also strongly-unlinkable.

▶ **Proposition 21.** *If an RRC scheme* R *is strongly-pseudorandom then it is also strongly-unlinkable.*

We now turn to present several ways to augment our basic RRC schemes so that they achieve strong-pseudorandomness, and hence strong unlinkability.

6.2 Constructing Strongly-Pseudorandom RRCs

We now present a way to turn our lattice-based constructions of RRC schemes to ones that provide strong pseudorandomness, and hence strong unlinkability. We start by describing such a mechanism for our LWE-based scheme, and then discuss how the same ideas can also be applied to our Ring-LWE-based scheme.

Immunizing our LWE-based RRC scheme $\mathsf{R}_{\mathsf{LWE}}$ against adversarial re-randomizations per the Definition 19 amounts to defining the beacon distribution D, the algorithms $\mathsf{R}_{\mathsf{LWE}}.\mathit{Precommit}$ and $\mathsf{R}_{\mathsf{LWE}}.\mathit{Extract}$, and the set $\mathcal{G}$ of admissible random strings. We do so as follows:

- D is the uniform distribution over $\{-1, 1\}^{m\times m}$.
- $\mathsf{R}_{\mathsf{LWE}}.\mathit{Precommit}(pp; r)$: the randomness r to the algorithm is parsed as a tuple $(\boldsymbol{R}, r')$ of a uniformly-random matrix $\boldsymbol{R}$ in $\{-1, 1\}^{m\times m}$ and randomness r' to a standard (not re-randomizable) statistically-binding non-interactive commitment scheme $\mathsf{C} = (\mathsf{C}.\mathit{Setup}, \mathsf{C}.\mathit{Commit})$ (for definitions of standard commitment schemes, see for example [17]). The algorithm then commits to $\boldsymbol{R}$ using C: it computes $\mathsf{pcom} \leftarrow \mathsf{C}.\mathit{Commit}(pp_{\mathsf{C}}, \boldsymbol{R}; r')$ and outputs pcom (the public parameters pp_{C} for C are sampled by the $\mathsf{C}.\mathit{Setup}$ algorithm during the operation of $\mathsf{R}_{\mathsf{LWE}}.\mathit{Setup}$ and are included as part of the public parameters of $\mathsf{R}_{\mathsf{LWE}}$).
- $\mathsf{R}_{\mathsf{LWE}}.\mathit{Extract}(pp, r, \mathsf{rand})$ parses r as $(\boldsymbol{R}, r')$ and treats rand as a matrix $\boldsymbol{R}'$ in $\{-1, 1\}^{m\times m}$. It outputs $\boldsymbol{R}'' \leftarrow \boldsymbol{R} + \boldsymbol{R}' \in \mathbb{Z}_q^{m\times m}$.
- The set $\mathcal{G} = \mathcal{G}(\mathsf{pcom}, r = (\boldsymbol{R}, r'), \mathsf{rand} = \boldsymbol{R}')$ is then the singleton set $\{\boldsymbol{R} + \boldsymbol{R}'\}$ if $\mathsf{pcom} = \mathsf{C}.\mathit{Commit}(pp_{\mathsf{C}}, \boldsymbol{R}; r')$. Otherwise, if $\mathsf{pcom} \neq \mathsf{C}.\mathit{Commit}(pp_{\mathsf{C}}, \boldsymbol{R}; r')$ then $\mathcal{G} = \emptyset$ and there is no admissible randomness. That is, $\mathcal{G}$ "checks" if pcom is a valid commitment to $\boldsymbol{R}$ given the randomness used to generate it, and if so, the only admissible randomness for $\mathsf{R}_{\mathsf{LWE}}.\mathit{Randomize}$ is the sum of $\boldsymbol{R} + \boldsymbol{R}'$.

We denote the RRC scheme obtained by these augmentations by $\mathsf{R}^{+}_{\mathsf{LWE}}$. We first argue that the scheme is correct per Definition 18. Condition 2 of the definition holds trivially. To see why Condition 1 holds, observe that honest r_is used for re-randomization are now m-by-m matrices, whose coordinates are independently sampled from a distribution which attains 0 with probability 1/2, and -2 or 2 with probability 1/4 each. A straightforward adaptation of the proof of Lemma 6 shows that it still applies (with a slightly worse constant C) and hence the previous proof of correctness still goes through.

As for security, the following theorem, proved in the full version, proves that $\mathsf{R}^{+}_{\mathsf{LWE}}$ satisfies strong pseudorandomness. In conjunction with Proposition 21, this implies that it is also strongly-unlinkable.

▶ **Theorem 22.** *The scheme* $\mathsf{R}^{+}_{\mathsf{LWE}}$ *is a strongly-pseudorandom RRC scheme.*

Strong unlinkability from the Ring LWE assumption. We can use a similar technique in order to augment our Ring-LWE-based RRC scheme with strong unlinkability. The only difference is that now $\boldsymbol{R}$ and $\boldsymbol{R}'$ are sampled as matrices of "short" polynomials. That is, the distribution D samples a matrix $\boldsymbol{R}'$ as follows: Each coordinate is an independent polynomial, whose coefficients are sampled independently and uniformly from $\{-1, 1\}$. *Precommit* samples a commitment to a matrix $\boldsymbol{R}$ sampled from the same distribution, and *Extract* outputs $\boldsymbol{R}+\boldsymbol{R}'$. Finally, the set $\mathcal{G}(\mathsf{pcom}, r, \mathsf{rand}) = \{\boldsymbol{R}+\boldsymbol{R}'\}$ as before if the precommitment pcom is consistent with r and $\emptyset$ otherwise. Correctness follows similarly as in the LWE case, replacing the use of Lemma 6 with Lemma 7. For strong pseudorandomness, we replace the use of the leftover hash lemma [29] with Lemma 4.[3]

6.3 Strong Pseudorandomness without A Randomness Beacon

The above approach requires a randomness beacon, which is a very reasonable assumption in the context of SSLE protocols. However, there might be other scenarios in which one might want to use RRCs without assuming the availability of such a beacon. This is formally captured by the above definitions by fixing D to be the constant distribution outputting $\perp$ with probability 1. In the full version, we present three different approaches to augment our schemes to provide strong unlinkability without assuming a randomness beacon.

References

1 Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (h)ibe in the standard model. In *Advances in Cryptology – EUROCRYPT 2010*, pages 553–572, 2010.

2 Martin R. Albrecht, Valerio Cini, Russell W. F. Lai, Giulio Malavolta, and Sri AravindaKrishnan Thyagarajan. Lattice-based snarks: Publicly verifiable, preprocessing, and recursively composable. In *Advances in Cryptology – CRYPTO 2022*, pages 102–132, 2022.

3 Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. *Theory of Computing Systems*, 48:535–553, 2011.

4 Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104, Dallas, TX, USA, October 31 – November 2 2017. ACM Press. `doi:10.1145/3133956.3134104`.

5 Prabhanjan Ananth, Apoorvaa Deshpande, Yael Tauman Kalai, and Anna Lysyanskaya. Fully homomorphic NIZK and NIWI proofs. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 356–385, Nuremberg, Germany, December 1–5 2019. Springer, Heidelberg, Germany. `doi:10.1007/978-3-030-36033-7_14`.

6 Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed Σ-protocol theory for lattices. In *Advances in Cryptology – CRYPTO 2021*, pages 549–579, 2021.

7 Sarah Azouvi and Daniele Cappelletti. Private attacks in longest chain proof-of-stake protocols with single secret leader elections. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, AFT '21, pages 170–182, 2021. `doi:10.1145/3479722.3480996`.

8 Sarah Azouvi, Patrick McCorry, and Sarah Meiklejohn. Betting on blockchain consensus with fantomette, 2018. `arXiv:1805.06786`.

9 Michael Backes, Pascal Berrang, Lucjan Hanzlik, and Ivan Pryvalov. A framework for constructing single secret leader election from MPC. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part II*, volume 13555 of *LNCS*, pages 672–691, Copenhagen, Denmark, September 26–30 2022. Springer, Heidelberg, Germany. `doi:10.1007/978-3-031-17146-8_33`.

10 Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafael del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In *Advances in Cryptology – CRYPTO 2018*, pages 669–699, 2018.

[3] Technically speaking, we require a generalization of Lemma 7, in which the coefficients of each entry of $\boldsymbol{R}$ may be chosen from different (but small) sets. Fortunately, the proof of 7 readily extends to this setting.

11 Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, , and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018. URL: `https://eprint.iacr.org/2018/046`.

12 Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology – EUROCRYPT 2019*, pages 103–128, 2019.

13 Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography*, pages 31–60, 2016.

14 Fabrice Benhamouda, Stephan Krenn, Vadim Lyubashevsky, and Krzysztof Pietrzak. Efficient zero-knowledge proofs for commitments from learning with errors over rings. Cryptology ePrint Archive, Report 2014/889, 2014. URL: `https://eprint.iacr.org/2014/889`.

15 Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2025–2038, Virtual Event, USA, November 9–13 2020. ACM Press. `doi:10.1145/3372297.3417893`.

16 Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In *AFT '20*, pages 12–24. ACM, 2020. Available online at eprint/2020/025.

17 Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography, Draft 0.6*. Cambridge University Press, 2023.

18 Dario Catalano, Dario Fiore, and Emanuele Giunta. Efficient and universally composable single secret leader election from pairings. Cryptology ePrint Archive, Report 2021/344, 2021. URL: `https://eprint.iacr.org/2021/344`.

19 Dario Catalano, Dario Fiore, and Emanuele Giunta. Adaptively secure single secret leader election from ddh. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC'22, pages 430–439, 2022. `doi:10.1145/3519270.3538424`.

20 Rutchathon Chairattana-Apirom and Anna Lysyanskaya. Compact cut-and-choose: Boosting the security of blind signature schemes, compactly. Cryptology ePrint Archive, Paper 2022/003, 2022. URL: `https://eprint.iacr.org/2022/003`.

21 Miranda Christ, Valeria Nikolaenko, and Joseph Bonneau. Leader election from randomness beacons and other strategies, 2022. URL: `https://a16zcrypto.com/posts/article/leader-election-from-randomness-beacons-and-other-strategies`.

22 Nuria Costa, Ramiro Martínez, and Paz Morillo. Lattice-based proof of a shuffle. In *FC 2019: Financial Cryptography and Data Security*, pages 330–346, 2019.

23 Justin Drake. Low-overhead secret single-leader election, 2019. URL: `https://ethresear.ch/t/low-overhead-secret-single-leader-election/5994`.

24 Luciano Freitas, Andrei Tonkikh, Adda-Akram Bendoukha, Sara Tucci-Piergiovanni, Renaud Sirdey, Oana Stan, and Petr Kuznetsov. Homomorphic sortition – single secret leader election for pos blockchains. Cryptology ePrint Archive, Paper 2023/113, 2023. URL: `https://eprint.iacr.org/2023/113`.

25 Chaya Ganesh, Claudio Orlandi, and Daniel Tschudi. Proof-of-stake protocols for privacy-aware blockchains. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 690–719, Darmstadt, Germany, May 19–23 2019. Springer, Heidelberg, Germany. `doi:10.1007/978-3-030-17653-2_23`.

26 Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 197–206, 2008. `doi:10.1145/1374376.1374407`.

27 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. URL: `https://eprint.iacr.org/2017/454`.

28 Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, , and Riad S. Wahby. Brakedown: Linear-time and post-quantum snarks for R1CS. Cryptology ePrint Archive, Paper 2021/1043, 2021. URL: `https://eprint.iacr.org/2021/1043`.

29 Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

30 George Kadianakis. Whisk: A practical shuffle-based ssle protocol for ethereum, 2022. X.

31 Thomas Kerber, Aggelos Kiayias, Markulf Kohlweiss, and Vassilis Zikas. Ouroboros crypsinous: Privacy-preserving proof-of-stake. In *2019 IEEE Symposium on Security and Privacy*, pages 157–174, San Francisco, CA, USA, May 19–23 2019. IEEE Computer Society Press. `doi:10.1109/SP.2019.00063`.

32 Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. In *Advances in Cryptology – CRYPTO 2022*, pages 71–101, 2022.

33 Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23, French Riviera, May 30 – June 3 2010. Springer, Heidelberg, Germany. `doi:10.1007/978-3-642-13190-5_1`.

34 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: `http://bitcoin.org/bitcoin.pdf`.

35 Ngoc Khanh Nguyen and Gregor Seiler. Practical sublinear proofs for r1cs from lattices. In *Advances in Cryptology – CRYPTO 2022*, pages 133–162, 2022.

36 Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016. Available online at eprint/2015/939.

37 Mayank Raikwar and Danilo Gligoroski. Sok: Decentralized randomness beacon protocols. In *Australasian Conference on Information Security and Privacy*, pages 420–446. Springer, 2022. available here.

38 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93, Baltimore, MA, USA, May 22–24 2005. ACM Press. `doi:10.1145/1060590.1060603`.

39 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009. Available online here.

40 Antonio Sanso. Towards practical post quantum single secret leader election (ssle) - part 1, 2022. X.

41 Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134, Santa Fe, NM, USA, November 20–22 1994. IEEE Computer Society Press. `doi:10.1109/SFCS.1994.365700`.

42 Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 617–635, Tokyo, Japan, December 6–10 2009. Springer, Heidelberg, Germany. `doi:10.1007/978-3-642-10366-7_36`.

Liquidity Management Attacks on Lending Markets

Alireza Arjmand
University of Alberta, Edmonton, Canada

Majid Khabbazian
University of Alberta, Edmonton, Canada

Abstract

Decentralized Finance (DeFi) continues to open up promising opportunities for a broad spectrum of users, with lending pools emerging as a cornerstone of its applications. While prominent platforms like Compound and Aave maintain a large share of the funds in lending pools, numerous other smaller pools also exist. Many of these smaller entities draw heavily from the design principles of their larger counterparts due to the complex nature of lending pool design.

This paper asserts that the design approaches that serve larger pools effectively may not necessarily be the most beneficial for smaller lending pools. We identify and elaborate on two liquidity management attacks, which can allow well-funded attackers to exploit specific circumstances within lending pools for personal gain. Although large lending pools, due to their vast and diverse liquidity and high user engagement, are generally less vulnerable to these attacks, smaller lending protocols may need to employ specialized defensive strategies, particularly during periods of low liquidity. We also show that beyond the six leading lending protocols, there exists a market value exceeding $1.75 billion. This considerable sum is dispersed among over 200 liquidity pools, posing a potentially attractive target for bad actors.

Furthermore, we evaluate existing designs of lending pools and suggest a novel architecture that distinctly separates the liquidity and logic layers. This unique setup gives smaller pools the adaptability they need to link with larger, well-established pools. Despite encountering certain constraints, these emerging pools can leverage the considerable liquidity from larger pools until they generate sufficient funds to form their own standalone liquidity pools. This design cultivates a setting where multiple lending pools can integrate their liquidity components, thus encouraging a more diverse and robust liquidity environment.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Lending Pools, DeFi, Interest Rate, Liquidity Management Attack

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.27

1 Introduction

Decentralized Finance (DeFi) protocols offer a solid foundation for financial investors seeking to earn returns on their assets in a decentralized manner. Lending and borrowing, one of the oldest financial applications, is transformed by blockchains, enabling the creation of liquidity pools that consolidate lender funds and facilitate borrowing. This arrangement presents an appealing opportunity for both parties; lenders earn interest from the moment they contribute their funds to the liquidity pools, while borrowers are assured of paying a fair interest rate for their borrowed amount.

This paper primarily focuses on over-collateralized lending pools [4], where, after liquidity providers contribute their funds to the pool, borrowers can access these funds by offering collateral in other assets. The collateral amount must exceed the borrowed sum to allow the lending protocol to guarantee a return of funds to the liquidity providers. Should the collateral amount drop below a certain threshold, the collateral can be converted into the borrowed asset, incentivizing third parties to repay the liquidity providers in a process known as liquidation [24].

Despite experiencing a decline in 2022 [26], the lending markets continue to expand, amassing a Total Value Locked (TVL) in excess of $13.2b across a multitude of blockchains [10]. While dominant lending markets such as Compound [7, 18] and Aave [1, 2] maintain the bulk of this value, new lending protocols inspired by these major lending pools are constantly emerging, contributing novel capabilities to the application layer for users. To gain traction, these newer lending protocols need to incentivize users to entrust their funds to their platforms. This often requires competition with larger lending pools through attractive incentives such as higher interest rates and novel application layer opportunities.

A key part of any lending pool is its interest rate formula, which determines how much borrowers have to pay back based on what they borrow. The importance of this formula lies in its potential to encourage certain behaviors: (i) It should incentivize borrowing by decreasing interest rates when ample liquidity is available; (ii) It should attract external liquidity providers to participate in the protocol by elevating interest rates when a significant portion of the liquidity is borrowed; (iii) It should stimulate the retention of some liquidity in the pool, enabling providers to withdraw at any time. To encourage these behaviors, several recognized formulas/models are frequently used by lending protocols [15].

In the widely adopted model, lending pools implement high interest rates on borrowed funds when usage approaches 100%. Consequently, if this level of usage persists for an extended duration, borrowers will be subject to significantly increased fees compared to the norm. To address this issue, these lending protocols depend on diligent users who actively monitor the situation. These users are incentivized to inject funds into the pool when interest rates are high. However, if these users lack sufficient funds to effectively reduce the usage or if there is a delay in their actions, borrowers in the lending pools may suffer substantial losses due to the elevated interest rates.

In this paper, we focus on small lending pools that adopt similar models. We postulate that a malicious liquidity provider, owning a significant share of a liquidity pool's reserves, can manipulate other actors to align with certain conditions for their benefit, potentially causing harm to others. We demonstrate that the relative lack of substantial liquidity funds and centralized liquidity providers in these smaller pools can expose them to various threats. In particular, we make the contributions:

- **Liquidity Management Attacks:** To highlight the vulnerability of small lending pools, we present two different liquidity management attacks on these pools. Furthermore, we delve into a general strategy that could be implemented by an attacker with sufficient funds, highlighting the incentives for users and a long-term approach that could prove profitable for the attacker but detrimental to the ecosystem. We also evaluate potential mitigation as well as risks involved in launching the proposed attacks.
- **Liquidity Aggregator:** We present a model in which lending pools separate their liquidity layer from their logic layer. By this means, smaller lending pools can integrate their applications with larger lending pools, thereby enhancing their liquidity safeguards. In this model, lending pools can coexist in dependent or standalone modes, allowing the community to avoid scattering liquidity across numerous platforms.
- **Lending Protocol Data Extraction:** We gathered data from the six biggest lending pools. Even though they hold most of the TVL, it's important to note that there is still a considerable amount of value in the remaining lending pools. This could potentially make them targets for malicious users.

The rest of this paper is organized as follows, Section 2 provides necessary background information. Section 3 introduces the mathematical model that forms the basis for our discussions throughout the paper. Section 4 outlines the logic behind two types of liquidity management attacks we investigate and illustrates how malicious actors can manipulate economic principles to meet their goals. Section 5 presents a design proposal to bolster the security of emerging lending pools, especially those with limited overall liquidity. In Section 6, we analyze the total value locked in on-chain lending pools, focusing on the six largest protocols from various perspectives. Section 7 surveys related work in this field. Finally, in Section 8, we wrap up our discussions and suggest potential avenues for future research.

2 Background

In this section, we present the fundamental concepts necessary to comprehend the subsequent content of the paper.

2.1 Blockchains

Blockchains comprise numerous underlying nodes that disseminate transactions throughout the system using a Peer-to-Peer (P2P) network [20, 6]. Each transaction typically aims to uniquely alter the global state. Transactions are appended to the blockchain within blocks in each round, following a consensus algorithm that determines the transactions' inclusion and sequence.

2.2 Decentralized Finance (DeFi)

Ethereum [32] employs a Turing-complete language named Solidity, enabling users to deploy *smart contracts*. These contracts broaden user capabilities by facilitating the creation of decentralized applications, giving rise to DeFi applications [31]. At present, Ethereum employs the Proof of Stake (PoS) consensus algorithm, which designates a block builder each round to select the transactions' order, which is then subjected to voting by other block builders. Once a block is produced in each round, all users can sequentially execute each transaction within the Ethereum Virtual Environment (EVM) to ascertain the current global state. One distinctive feature of the EVM is that its operations are deterministic and atomic, altering the state only upon success. Therefore, given any pre-state and specific inputs, each node would produce identical outputs. These attributes, coupled with Ethereum's high throughput, have led to novel, transparent DeFi applications not traditionally found in Centralized Finance (CeFi) [23]. Furthermore, Ethereum's allowance for smart contract composability has resulted in the establishment of complex ecosystems.

DeFi has continued to thrive over the past year, attracting numerous users and boasting more than $41.5b in TVL. The absence of third parties and the transparency offered by DeFi applications make them an attractive prospect for many. Popular applications of DeFi include lending pools [4], Decentralized Exchanges [33], Yield aggregators [8], and stablecoins [19].

2.3 Attacks on DeFi

While code transparency is beneficial, it can also simplify the task of spotting faulty code. If such vulnerabilities are detected by attackers, they could lead to massive security breaches. In some of the most significant hacks, such as [22, 5], attackers exploited application layer bugs to siphon user funds. The classification of attack strategies has been thoroughly documented in the literature [35, 3, 14, 11], which is essential in assisting the community in identifying and avoiding patterns that could lead to undesirable consequences. Concurrently, there exist open-source libraries [21] that strive to provide secure building blocks for contracts. This enables protocol developers to ensure the safety of their code's foundational elements.

2.4 High frequency trading

Decentralized markets have given rise to on-chain high-frequency trading [9, 34]. This environment, while presenting many opportunities, also attracts malicious users aiming to seize on-chain opportunities by tampering with transaction ordering. Tactics such as front-running and sandwich attacks are used to drain funds or steal opportunities away from unsuspecting users. To mitigate this, private relayers such as Flashbots [12] have emerged. These entities promise users certain assurances about their transaction inclusion, thereby safeguarding them from generalized front-runners.

3 System model

In this section, we aim to formalize the actions of users who can impact a lending protocol. To simplify the analysis, we focus on a specific subset of actions in lending pools and disregard other activities such as liquidations and absorptions. We assume the presence of numerous users in the system. A user u in our system model is a tuple $u = (S, B, C)$, where S is the amount of fund the user has supplied to the protocol, B is the amount of funds borrowed by the user, and C is the total collateral the user provided to the protocol. For simplicity, in our model, we convert the values of S, B, and C to a common base value (e.g. USD).

The balance of a user $u_i = (S_i, B_i, C_i)$ is defined as $S_i - B_i$. If a user's balance is greater than zero, the user is considered a *liquidity provider*; otherwise, if its balance is less than zero, the user is identified as a *borrower*. A borrower must have adequate collateral in the system for the borrowed balance. Since liquidations are not factored into our model, the following condition should be true for each user u_i:

$$S_i + EC_i > B_i,$$

where EC_i is the effective collateral for each user, that is

$$EC_i = \Sigma_j c_{ij} \times f_j \times rate_{USD/j}$$

Here, f_j represents the collateral factor for each asset. We denote the total amount of each variable in the entire protocol using the "total" subscript, such as S_{total}.

In our system, the borrowers in the system are subject to an interest R calculated using the *kinked interest rate model* as follows:

$$R = \begin{cases} R_0 + R_{low} \times U & if\ U \leq kink \\ R_0 + R_{low} + R_{high} \times (U - kink) & if\ U > kink \end{cases} \tag{1}$$

Table 1 Terminology used in system model.

Character	Meaning
L	Supplied liquidity
B	Borrowed amount
R	Interest rate
U	Utilization
α	Attacker liquidity percentage
EC	Effective collateral
$kink$	Optimal utilization

In this formulation, U denotes the protocol's utilization, calculated as $\frac{B_{total}}{S_{total}}$, where $kink$ represents the optimal utilization rate, often referred to as the 'kink rate'. The terms R_0, R_{low}, and R_{high} signify the base interest rate, the lower slope for utilization, and the sharp increase in interest rates when utilization surpasses the kink rate, respectively. Borrowers are assumed to accrue interest with each passing block, adhering to this interest rate model:

$$Fee_i = R_U \times B_i \times t \tag{2}$$

We also assume that the protocol reserve doesn't accumulate any yields and all borrower fees are shared among the liquidity providers. To model the reserve, we can consider the reserve amount as one of the liquidity providers.

Collusion model. In the context of lending protocols, it is conceivable that a group of users may collude to achieve a common objective. Thus, we consider an adversary A who can compromise multiple accounts with cumulative supply of up to fraction α, such as:

$$\alpha \geq \frac{\Sigma_e S_e}{S_{total}} \tag{3}$$

Where α is the maximum fraction of overall funds that an attacker can control.

4 Attacks on lending markets

In this section, we examine the overarching structure of lending pools and present two forms of attacks that enable an adversary to impose specific conditions on the liquidity pool by employing economic strategies to secure a desired outcome. These outcomes could be:

- **More income:** An attacker can augment the fees extracted from other participants within the pool over a specific time frame.
- **Denial of Service:** An attacker can obstruct access to the rest of the participants, effectively preventing them from either borrowing or withdrawing their liquidity from the pool.

While these attacks pose potential complications for other users, they necessitate a substantial amount of liquidity from the attacker to fulfill the preconditions of launching the attack. Consequently, the attacker's risk level escalates in correlation with the growth of this prerequisite amount. The Compound and Aave protocol models are currently the most influential among the lending pools, widely implemented by smaller lending pools and occasionally forked from the main projects. Given the vast liquidity diversity and substantial user base of the top protocols with the highest TVL, an adversary would face a formidable task executing these attacks. However, the situation is different for smaller pools. Here, an attacker could instigate these attacks with a lower risk and initial capital, thereby realizing a profit. Thus, we demonstrate that smaller pools cannot merely replicate the strategies of larger entities. They must devise additional defence mechanisms against such attacks while their liquidity pool is relatively small, thereby safeguarding their liquidity providers and borrowers.

In the remainder of this section, we commence by elucidating the potential attacks and demonstrating how an attacker with sufficient liquidity can enforce other actors to comply with specific conditions. We then proceed with an analysis of the attacker's risk before deliberating on some design decisions that new lending pools should avoid.

4.1 Utilization kink attack

While borrowers secure funds by depositing an overcollateralized quantity of tokens in the protocol, they pay ongoing fees determined by the length of their loan. These fees fluctuate based on the degree of liquidity utilization, with adjustments made following each transaction processed by the protocol. Generally, it is anticipated that the borrowing rate maintains proportionality with the borrowed amount and the R_{low} delineated in the interest rate formula. However, when the utilization quantity exceeds a predetermined threshold or " kink", all borrowers become liable to pay supplemental fees to the liquidity providers. The objective of this kink value is to motivate all participants to act, thereby releasing liquidity within the protocol: (1) as a liquidity provider, the increased fees offer an incentive to contribute more liquidity from out of the protocol, and (2) as a borrower, the prospect of evading excessive fees incentivizes the repayment of the borrowed amount. Both actions lead to a decrease in total utilization and consequently a reduction in fees. By comparing the fees at maximum lending protocol utilization and at the kink value, we notice that in some protocols the fees can unexpectedly jump to more than ten times. This indicates that if an attacker were to elevate these values by either borrowing the rest of the remaining liquidity, or pulling out his own liquidity out of the protocol, they could compel borrowers to bear extensive fees. In such scenarios, smaller pools face two significant threats compared to their larger counterparts:

- **Lesser liquidity required:** Attackers need a smaller volume of liquidity to drive up fees, consequently exposing themselves to lower risks.
- **Smaller group of active users:** In such circumstances, the lending pool requires either active external liquidity providers or borrowers to regulate utilization. A smaller lending pool implies a lower number of participants monitoring such activities in the system, hence increasing the likelihood of such attacks.

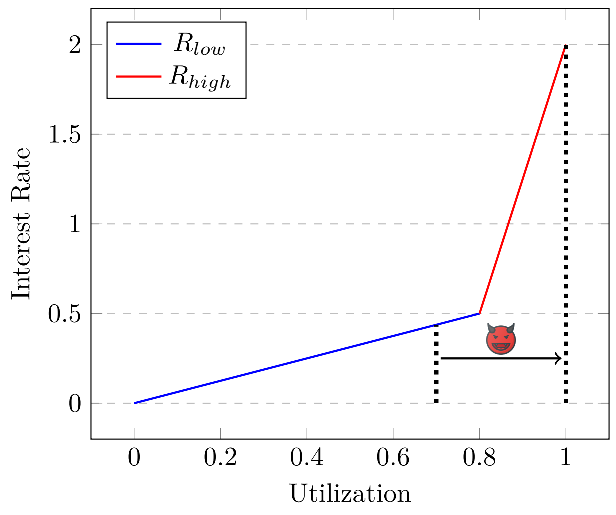


Figure 1 The kinked rate model can be exploited by an attacker through either increasing the utilization of the protocol by borrowing more or withdrawing funds.

4.1.1 Simplified attack

In order to exemplify this attack, we explore a hypothetical scenario involving a single liquidity provider, Alice, and a borrower, Bob. This analysis demonstrates how Alice can increase the utilization potentially to secure additional fees from Bob. Subsequently, real-world protocol figures are utilized to replace the formulas and estimate the possible damage an attacker can cause borrowers to pay.

Scenario Setup. Consider a lending platform characterized by parameters R_{low}, R_{high}, and $kink$, which are used to compute the interest rate. Initially, Alice contributes S initial funds to the protocol. Subsequently, Bob borrows an amount B, setting the protocol's utilization at the $kink$ amount by offering C in collateral value with collateral factor f.

Attack Execution. Alice currently receives fees from Bob proportionate to $kink * R_{low}$. Nonetheless, Alice can elevate the utilization by opting for one of the following strategies to increase the protocol's utilization:

- She may withdraw $(1 - kink) * S$ liquidity from the protocol.
- She might borrow the remaining amount of $(1 - kink) * S$ and pay those fees to herself, since she is the sole liquidity provider. In this case, Alice needs more funds comparing to the previous method to borrow and execute the attack.

Any of these actions would surge the protocol utilization to 100%, thereby significantly escalating Bob's fee. We can calculate the Bob's new fee, which is proportionate to $kink * R_{low} + (1 - kink) * R_{high}$. We can see that Bob needs to pay $1 + \frac{(1-kink)*R_{high}}{kink*R_{low}}$ times more fees.

Aftermath. Although Bob retains the option to stop this attack at any point by repaying his borrowed positions, he remains accountable for fees corresponding to the duration he borrowed the funds from the protocol. Nevertheless, Bob's response may be hindered for various reasons:

- He may not have enough liquidity to repay the borrowed sum, especially if these funds have been invested and locked elsewhere.
- He may be offline or negligent in monitoring the protocol's fees.

Furthermore, many protocols accumulate fees for borrowers in a manner that escalates their borrowing position over time. This means that by exploiting these circumstances, Alice not only forces Bob to endure higher fees but could also cause the liquidation of his position if the accumulated fees surpass Bob's initial estimations. Bob's position can even get liquidated if the following formula becomes true:

$$EC_{Bob} < B + fee \tag{4}$$

While Bob may have provided ample collateral to cover the protocol's standard fees, Alice could potentially elevate Bob's fees, leading to the liquidation of his position and opening up another potential profit source.

Numerical example. As a straightforward example, consider a lending pool emulating the interest rate parameters of Compound V2's cETH contract. As of this writing, this contract has an R_{high}/R_{low} ratio of 217.78 and a kink value of 0.8. Consequently, for utilization rates exceeding 80 percent, we observe a significant increase in the fees taken from borrowers. Yet, Compound V2 is a well-known contract, frequently monitored by numerous users. In contrast, for newly generated contracts which are copying these values, the utilization kink attack can present a genuine threat. An attacker could amplify fees by escalating utilization from 80 to 100 percent, by $((1-0.8)/0.8) \times 217.78 = 54.445$ times. Thus, if Alice successfully executes this attack against Bob for merely a single day, the profits generated would approximate those accrued from nearly two months of honest investment.

4.1.2 Utilization kink attack in general setting

While the prior example was a basic version of the attack with just two actors in the system, it served to illustrate that such attacks are indeed possible. However, in real-world situations, the number of actors, including both honest users and adversaries, is typically greater than one. In this section, we aim to shape a scenario involving multiple actors, where adversaries might work together to conduct the explained attack on a specific lending pool.

Collusion among liquidity providers. In order to examine the attack in a broader context, we need to account for realistic interactions among actors. In this section, we concentrate on a specific scenario where attackers could potentially enhance the utilization rate by withdrawing their available liquidity. To simplify this without compromising the mathematical validity of our analysis, we assume that a fraction, represented as α, of all liquidity provided to the pool is controlled by colluding adversaries. In this system, where $1-\alpha$ represents honest participants, the adversaries decrease their shares by withdrawing their funds. Interestingly, under certain conditions met by the interest rate formula, attackers could increase their fees even after reducing their shares. One approach for adversaries to collude atomically, would be through a smart contract. The progression of steps is outlined below:

1. Any adversary could deploy an attack smart contract, equipped with three key functionalities: (1) obtaining permission from users to manage their liquidity tokens, (2) withdrawing funds from each adversary's account to increase the utilization while reducing their respective shares, and (3) returning funds to the liquidity pool if the liquidity kink attack ceases to be profitable.
2. Each adversary could then grant a certain amount of liquidity provider tokens to the deployed contract using the pool's functions, permitting the contract to manage liquidity on behalf of each adversary.
3. Once all permissions are received, a specific threshold of signatures from adversaries could initiate the event of pulling liquidity from the protocol to boost utilization.
4. At this point, adversaries can monitor on-chain events to assess the profitability of the lending pool.
5. Should a new honest liquidity provider join the lending pool, or borrowers repay their borrowed amounts to an extent that it no longer remains profitable for attackers to withhold their funds, they can refund all the liquidity and revert to the initial state.

This strategy enables adversaries to minimize liquidity management risks and, in the worst-case scenario, return to the starting state. By providing adequate permissions, adversaries can utilize the attack contract to impose higher fees when feasible.

Scenario Setup. In this particular situation, we presume that attackers are already in possession of α percent of the total liquidity pool, denoted as L. The borrowed amount is represented by B. The kinked model, which we discussed earlier, guides the calculation of the interest rate. Moreover, we operate under the assumption that the attackers have already initiated the attack contract and have authorized it to either deposit or withdraw funds as necessary. We assume that prior to the attack, the utilization U is less than the kink value. We also assume that attackers possess sufficient liquidity to elevate the protocol's utilization above the kink value. If they lack this amount, the attack would be ineffective and they would merely diminish their own shares. Finally, we operate under the assumption that all fees derived from borrowers are directed to the liquidity providers, with none retained by the protocol itself. This simplifying assumption aids in streamlining the model, though in real-world applications, a portion of the fees is typically allocated to a community wallet managed by a DAO or an admin. Should the attackers choose to retain all their funds within the liquidity pool, behaving honestly, the fees they would receive would equate to the following amount:

$$fee_{honest} \propto (R_0 + \frac{B}{L} * R_{low}) * \alpha \tag{5}$$

Attack Execution. For attackers to boost the utilization, they initially need to calculate the exact amount of funds, termed as x, to withdraw from the protocol to yield higher fees. We assume that when attackers extract this x amount from the protocol's reserves, it drives the utilization beyond the kink value. As a consequence, the fees that would then accrue to the attackers can be computed as follows:

$$fee_{attack} \propto (R_0 + R_{low} \times kink + ((\frac{B}{L-x}) - kink) \times R_{high})(\alpha - \frac{x}{L}) \tag{6}$$

In the preceding equation, the attackers' shares drop from α to $\alpha - x/L$. Simultaneously, the total amount of funds in the protocol diminishes by x, though the borrowed amount remains unchanged.

Our objective is to pinpoint the ideal amount that adversaries should extract from the protocol to maximize fee_{attack}. We attain this by identifying the global maximum obtained from the function's derivative. The solution to this is realized when the condition $dfee_{attack}/dx = 0$ is fulfilled, the optimal amount can be determined by solving the following equation:

$$\frac{B \times R_{high} \times (a - \frac{x}{L})}{(L-x)^2} = \frac{R_{high} \times (\frac{B}{L-x} - U) + R_{low} \times U + R_0}{L} \tag{7}$$

This, naturally, would be the ideal value according to the condition if it lies within the range $x < L - B$, and $x > kink * L - B$.

Risks. Even though attackers stand to profit while the utilization remains high, they are simultaneously accepting certain risks. We explore these primary risks in this section.

- **Borrower Attrition:** By initiating the utilization kink attack, attackers risk compromising their long-term income. Specifically, they may incentivize borrowers to withdraw their money, potentially redirecting it to other protocols. Consequently, a lending pool subject to such attacks may fail to instill trust in new borrowers. Nonetheless, an attacker could easily shift their funds to other protocols, given there are multiple that offer such services.
- **Monitoring Challenges:** The preceding section demonstrated that certain conditions need to be met for a profitable scenario. Given these conditions may change as new actors join and leave the system, attackers can respond quickly when the situation ceases to be profitable. Failure to do so could result in a loss of potential fees that could have been earned through honest investing.

- **Security Considerations:** Participating in a protocol implies that users, both honest and dishonest, trust the protocol to be secure. However, there's always a risk that a protocol may contain a bug leading to a loss of all funds. When an attacker moves between protocols to execute liquidity management attacks, they are inherently trusting these protocols not to be compromised. If a breach does occur, they might lose all their funds.

Mitigation recommendations. The potential threat of liquidity kink attacks can be partially mitigated at the protocol's design phase, offering some level of protection to borrowers. One potential remedy involves demanding a commitment of liquidity from providers. The majority of honest liquidity providers aim to keep their resources in the market for an extended duration. In defense of borrowers, the protocol could stipulate a minimum time commitment from these providers, thereby inhibiting attackers from removing their funds and artificially increasing the protocol's utilization. An alternative could be the establishment of " fee tiers", whereby the protocol rewards providers who have pledged their resources over a longer time frame with higher fees. However, this strategy only stops attackers from withdrawing their funds, while the possibility of borrowing the remaining amount to amplify utilization still exists.

4.2 DoS attack on liquidators

When liquidity providers contribute funds to a protocol, it is generally assumed that sufficient funds will be available for regular withdrawals when needed. The portion of funds supplied to the protocol but not borrowed is typically eligible for withdrawal. However, it is crucial to acknowledge that this mechanism does not guarantee withdrawals, as it is incentivized by imposing fees on borrowers when the total protocol utilization exceeds the specified threshold (kink). Additionally, the fee mechanism is often time-based, considering the duration between borrow and repayment transactions to calculate the final fee. Consequently, if liquidity is borrowed and repaid within the same block, the borrower only needs to cover the gas fee and is not subject to additional fees from the protocol.

An adversary could exploit (1) the absence of guaranteed withdrawals and (2) borrow fees based on time, to launch a DoS attack. This attack could impact liquidity providers who are trying to withdraw their funds from many lending protocols, as well as borrowers attempting to secure a loan after providing sufficient collateral.

4.2.1 Simplified Attack

Here, we discuss a simple attack scenario, Suppose Alice is a liquidity provider in a lending protocol, supplying \$300,000 out of a \$1 million pool. The utilization level is currently at 70%, meaning \$300,000 of the pool remains available for both borrowers and liquidity providers to utilize. Alice urgently needs to withdraw the entire \$300,000 from the protocol. Bob, observing this, aims to prevent Alice's withdrawal opportunity. He already has sufficient collateral provided to the protocol and initiates two transactions: (1) a transaction with a higher gas fee than Alice's to front-run her transaction and borrow the entire \$300,000, resulting in 100% utilization, and (2) a transaction with a lower gas fee than Alice's to back-run her transaction and push the borrowed amount back into the protocol. By sandwiching Alice in this manner, Bob effectively denies her the withdrawal by causing her transaction to fail since there are no available free funds in the pool.

It is worth noting that in the above example, any other withdrawal requests from third parties would also fail since Bob has drained the protocol of funds. Furthermore, during this process, Bob would only pay the gas fees for the two transactions, which is a relatively small amount compared to the disruptive impact inflicted upon Alice within the system.

In addition to targeting specific users, an attacker can also attempt a generalized DoS attack against the entire network. In this scenario, the attacker aims to include one transaction at the beginning of a block and another transaction at the end of the same block. If successful, this strategy can effectively prevent anyone within the system from withdrawing funds from the protocol.

4.2.2 DoS attacks in general setting

In order for an adversary to launch DoS attacks on real-world systems, they require access to an amount of funds denoted as x. They can cause any withdrawal to fail if its size surpasses this threshold:

$$Withdrawal > L - B - x \tag{8}$$

Assuming that liquidity pools typically maintain utilization up to their optimal utilization, an attacker could disrupt any withdrawal provided they have access to $L * (1 - kink)$ funds. If the attacker's funds are already in the protocol as liquidity, they could withdraw their funds. Alternatively, if their funds are outside of the protocol, they could borrow the necessary amount temporarily for just one block. Given they can perform both these actions within a single block, they neither forfeit any income nor incur any fees. This is because the duration of the liquidity withdrawal or borrowing within the same block is effectively zero.

Risks. To execute a Denial of Service attack on users submitting transactions to a public mempool, an attacker can attempt to accomplish this objective by sending one transaction with a higher gas price and another transaction with a lower gas price. However, there is a risk involved as these transactions may not be included in the desired block. To mitigate this risk, an attacker can minimize the issue by bribing block builders within the blockchain network, requesting them to include all the target transactions in their subsequent block. By doing so, the attacker's risk exposure would be reduced. Alternatively, the attacker can opt to send transactions to a private relayer, such as flashbots, which ensures the " next-block-or-never" attribute. This approach allows the attacker to bundle the user's transactions into a meticulously constructed bundle and transmit it to the private relayer. In cases where an attacker is unable to successfully execute sandwich attacks on their target, their transactions remain valid and can be processed on the network. Hence, they might incur borrowing fees over several blocks, which could be a considerable amount given that the utilization is boosted to 100 percent, and the borrowed sum is substantial.

Mitigation recommendations. To effectively mitigate such attacks, implementing protocol-level measures is crucial. It is important to acknowledge that the DoS attack described does not incur a protocol-level fee, making it relatively inexpensive for an attacker to execute. One effective mitigation strategy is to introduce a percentage-based fee within the borrowing process. This means that when a user borrows a certain amount, they would be required to pay a fee calculated as follows:

$$Fee_i = R_U \times B_i \times t + B_i \times proportionalFee \tag{9}$$

By implementing this approach, the cost for an attacker to execute a DoS attack would increase proportionally with the size of the borrowed amount. As the attacker needs to deplete the remaining funds in the pool, the associated cost becomes significant, acting as a deterrent for such attacks. Furthermore, users can proactively protect themselves against these attacks by opting to send their transactions through a private relayer. This approach helps safeguard users from becoming targets of DoS attacks orchestrated by the attacker. However, it is important to note that these solutions may not be effective against the generalized DoS attacks previously discussed.

4.3 Economical games by adversary

In the present analysis, an attempt is made to envision the potential tactics of an adversary within the domain of lending pools to gain profits over an extended period. There are several incentives that may prompt adversaries to initiate such maneuvers, which are discussed in the ensuing sections:

- **Profit Realization:** The most straightforward objective for an adversary could be to accumulate profits. In the event an adversary consistently executes a kink utilization attack, they could potentially accrue multiple rounds of rewards. However, repeated instances of such attacks may compel borrowers to discontinue using the protocol.
- **Control over Access:** By leveraging a DoS attack, adversaries could exercise control over the liquidity providers' access to their funds. In theory, adversaries may be able to immobilize users' funds. However, in practice, it is more possible to cause delays in withdrawals from the protocol resulting in weak censorship [29]. Such delays can prove critical, particularly during periods of financial instability [30].
- **Attrition of Protocol Users:** A possible adversary objective could be to deter users from engaging with a specific protocol. If the adversary's liquidity is sizable in comparison to the entire pool, by performing such attacks, they could result in actors blacklisting the protocol. This is feasible through two mechanisms, for liquidity providers, they may join the protocol when they observe a spike in utilization but as the attacker re-infuses funds, utilization and consequently fees drop. Borrowers, on the other hand, may be subjected to substantially higher fees frequently, making the protocol a less attractive option.

An attacker can meticulously plan and execute such attacks over an extended duration following several steps:

1. Firstly, the attacker must amass significant funds, either through their own capital or via colluding with other adversaries.
2. Subsequently, they must identify vulnerable protocols with a small liquidity pool, relative to their initial funds.
3. Initial investment in the protocol may be conventional, followed by an inflow of investment which reduces the overall fees paid by borrowers. This leads to a situation where other liquidity providers exit the protocol in pursuit of higher returns elsewhere, or more borrowers enter the pool. The attacker must wait until their share is significantly higher than the remaining liquidity to borrow in the protocol, a stage that may occur over an extended period, such as a week. During this time, adversaries earn interest at a standard rate.
4. Once utilization has risen and remaining liquidity is considerably lower than the adversaries' shares, attacks can be launched to achieve their objectives. This stage should ideally be of a short duration since the execution of a utilization kink attack incentivizes other actors to balance utilization. Attackers can respond by further reducing their position upon other actors' actions, thereby continuing to accrue interest. If a large liquidity provider enters the system, attackers can reinfuse all withdrawn funds back into the protocol to sustain fee earnings. However, honest liquidity providers might have no incentive to aid a pool under attack if they anticipate temporary high utilization, making it unadvisable for them to move large volumes of liquidity to help the pool.

5. Continued attacks may lead to general actors in the network blacklisting the attacked protocol, in such situations attackers can easily migrate to a new vulnerable protocol.

In this economic game, attackers stand to profit over the long term. Two primary issues arise:

- **Low-Risk, High-Reward Game for Attackers:** Attackers stand to gain exponentially from 5 to 50 times more fees during the attack period without facing any substantial risks unless the protocol experiences a major hack. This allows them to perpetuate such activities over a long duration.
- **No Financial Incentives for Honest Players:** Existing pools incentivize players by raising interest rates; however, if attackers respond swiftly to honest actors joining the pool, there would be no financial incentive for honest players to rescue minor protocols.

Hence, protocols need to address these attacks at the design level to foster growth and safeguard their users against malicious activities.

While it is feasible for an attacker to simultaneously execute the mentioned attacks by elevating the utilization to its maximum, the objectives for conducting each attack differ. Here, we discuss some of these variations:

- **Utilization Kink Attack:** To execute this attack, malicious liquidity providers need to initially supply liquidity to a specific pool and wait until a part of their liquidity is borrowed. Only then can they employ the remainder of their funds to increase the utilization. In such attacks, all borrowers within the pool are targeted, and the attacker's profit accumulates over time.
- **DoS Attack:** In order to carry out a DoS attack, attackers can retain their funds outside the protocols, monitor multiple systems, and potentially target specific actors if their funding is sufficient. A DoS attack is intended to transpire swiftly within a specific block and is not a continuous action. This approach aims to avoid associated fees.

5 Liquidity aggregation

In previous discussions, we explored the issue of liquidity attacks. We proposed some tactical solutions, like extending liquidity commitments and setting base fees, to deal with such issues. But in this segment, our aim is to get to the core of the problem and offer a comprehensive solution. Our solution could safeguard new lending pools from potential attacks while facilitating their rapid growth.

Often, smaller lending pools try to emulate the larger ones such as Compound and Aave. This leads many protocols to design their logic layer centered around their liquidity pool. In this setup, the logic and liquidity components become inseparable parts of a single, large project. Consequently, each pool has to grow independently. Our proposition is to separate the liquidity and logic layers in the design of such protocols. This separation could let several protocols combine their liquidity layers, possibly strengthening the weaker pools. We recommend the following three-step launch for every new liquidity pool:

1. Design the pool such that the logic and liquidity layers are separate. The logic layer should only interact with the liquidity layer when necessary. This arrangement could allow the liquidity layer to be shared among many protocols.
2. Initially, smaller liquidity pools can connect themselves to larger pools such as Compound. This connection means that they only run out of liquidity when Compound does, protecting them from most liquidity management attacks. This method enforces some limitations on the smaller pool, as it has to conform to the larger pool's constraints.
3. Once the connected pool has sufficient funds, it can operate independently and set its own rules.

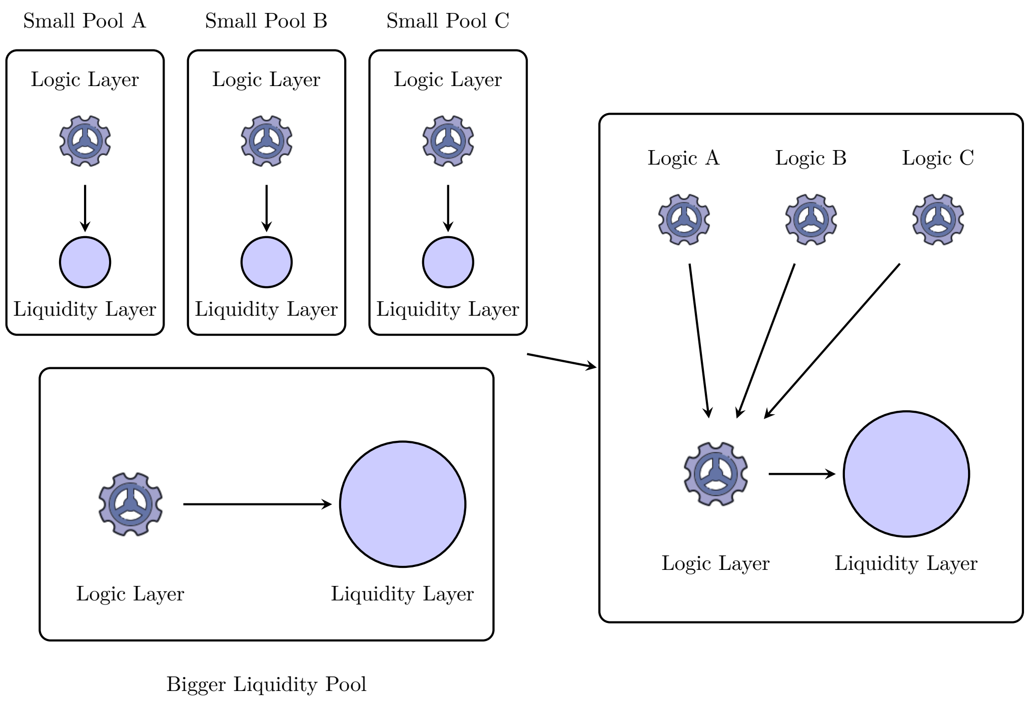


Figure 2 Liquidity aggregation process, how smaller pools can piggyback off larger pools.

By following these steps (as shown in Figure 2), an ecosystem of lending pools can reap mutual benefits. These benefits include:

- **Attack Resilience:** Smaller pools protect their users from attacks. It becomes more difficult for an attacker to raise borrowers' fees. Also, liquidity providers have the freedom to withdraw their funds at any time since the larger underlying pool provides more liquidity.
- **Larger Shared Pool:** The larger pools also benefit from this arrangement. They now have a larger pool of liquidity providers. Many protocols can use their liquidity for security, while merging their pools to enhance the overall security of the ecosystem.

In the following parts of this section, we aim to explain the complexity in the process of implementing such systems.

5.1 Designing Logic and Liquidity Layers

The goal of this section is to propose a design that separates the logic and liquidity layers of a lending pool. However, we still need these layers to merge together and form a complete lending system. This design expands upon the traditional lending pools' design of one-to-one logic and liquidity layers. It also potentially allows for the integration of multiple logic layers without the need to change the implementation of the liquidity layer.

The logic layer of the lending protocol is deployed via a smart contract, which should be the point of interaction for all users of the protocol. This means the logic layer must handle all bookkeeping and monitor each participant's activity, and it is not designed to hold any funds. When users interact with the protocol via the logic layer, it facilitates the transfer of funds between users and the liquidity pool after conducting necessary checks. On the other side, the liquidity layer, which holds all funds, should only respond to the logic layer contract.

A design layer should have the capability to (1) interface with another logic layer, thereby piggybacking on the infrastructure of another protocol, or (2) function as a standalone liquidity layer, in which it independently manages all of its funds.

5.1.1 Piggybacking Liquidity Pool

When a design layer is in piggybacking mode, it is connected to another design layer. This allows us to establish a system like $D_1, D_2, \ldots, D_N, LL_N$, where D_is are design layers and LL_N is the liquidity layer that only responds to D_N. Here, $D_1, D_2, \ldots, D_{N-1}$ are all in piggybacking mode, and D_N operates in standalone mode. While users can interact with any of the D_i to use their services, their liquidity will be forwarded through $D_i + 1, D_N$ and must comply with all their logic. In this setup, each of D_i has its own users, but all that D_{i+1} sees from the previous logic layer is the entry of D_i, which is using the system just like other users. The simplest version of the use case that interests us is where $N = 2$. Here, D_1 is a small lending pool, and D_2 is one of the largest existing lending pools, such as Compound. In this setting, while users interact with the D_1, their funds are getting accumulated in D_2's pool LL_2. The significant benefit here is that if D_1 runs out of funds, it is backed up by the bigger lending pool's funds and can support its users. We delve deeper into how each basic functionality changes when the design layer is piggybacking off other design layer when a user interacts with D_1:

- **Supply:** Whenever a user supplies amount X to the D_1, then supply of the system changes as:

$$\begin{aligned} S_{D1,user} &\mathrel{+}= X \\ \forall_{1<i\leq N} S_{Di,Di-1} &\mathrel{+}= X \\ L &\mathrel{+}= X \end{aligned} \tag{10}$$

  This means that each logic layer supplies funds to the next one, and the final pool supplies it to the pool.

- **Collateral:** when users supply collateral to the protocol, the state changes are similar to the supply:

$$\begin{aligned} C_{D1,user} &\mathrel{+}= X \\ \forall_{1<i\leq N} C_{Di,Di-1} &\mathrel{+}= X \\ C &\mathrel{+}= X \end{aligned} \tag{11}$$

- **Borrow and liquidation:** For a borrow of amount X to happen, the borrow process is happening in every single layer. Therefore, the collateral that user has provided, should follow the equation below:

$$X > max_i(\Sigma_c(C_{user,c,i} \times f_{c,i})) \tag{12}$$

  This implies that the collateral tokens submitted should exceed the borrowing amount in each logic layer. If the aforementioned condition is not met, the funds could potentially face liquidation in one of the layers. For protocols to ensure that the equation above is never broken, they need to limit their collateral factors, so that $f_{c,i} < f_{c,i+1}$. in such cases the collateral equation gets reduced to a limit against the effective collateral of the user at layer 1:

$$X > \Sigma_c(C_{user,c,1} \times f_{c,1}) = EC_{user,1} \tag{13}$$

The state changes for borrow are:

$$\begin{aligned} B_{D1,user} &\mathrel{+}= X \\ \forall_{1<i\leq N} B_{Di,Di-1} &\mathrel{+}= X \\ B &\mathrel{+}= X \end{aligned} \tag{14}$$

When a user seeks to borrow from the protocol and a layer runs out of liquidity, the protocol can borrow from the layer beneath it. This mechanism increases the confidence in liquidity availability.

- **Interest Rate Calculation:** Should there be no borrow at layer i, the total liquidity supplied to this layer, denoted as $S_{total,i}$, earns interest at the rate of the succeeding layer, or $i+1$. This follows the formula:

$$R_{i+1} \times S_{total,i} \tag{15}$$

Now, if any borrowing occurs from the protocol at layer i, the interest rate from the underlying protocol is given by:

$$R_{i+1} \times (S_{total,i} - B_{total_i}) + R_i \times B_{total_i} \tag{16}$$

Which depends on the interest rate of D_i. In order to incentivize more liquidity providers to join the protocol with an increase in borrowing, it is necessary that the condition $R_i \geq R_{i+1}$ be met. This requirement ensures that the previously mentioned formula progressively increases with the growth in borrowing positions. It indicates that the interest rate for layer i should surpass that of layer $i+1$. The proposed interest rate for level i extends from the kinked interest rate algorithm, following the subsequent equation:

$$\forall_{1\leq i<N}, R_i = \begin{cases} R_{i+1} + R_{low,i} \times U_i & if\ U \leq kink \\ R_{i+1} + R_{low,i} \times kink + R_{high,i} \times (U_i - kink) & if\ U > kink \end{cases} \tag{17}$$

The interest rate at each level is influenced by U_i. A significant difference in this model is that U_i can exceed the value of one. This is because each layer can lean on the next one for support, and therefore the borrowed amount within a specific protocol can go beyond the supplied amount. However, this also leads to a rise in the interest rate. To stop the growth of the interest rate at max utilization, protocol designers that are using this model could replace the U_i value with $Min(1, U_i)$.

In this setup, the outermost design layers can make use of the liquidity from all underlying protocols. However, this comes at the cost of stricter restrictions on their protocol variables. This implies that for an attacker to carry out a DoS attack on layer i, they now need to have enough funds to exhaust all layers from $i+1$ to N. On the other hand, if a lending protocol wants to connect to another protocol's logic layer, they don't need to set a steep $R_{high,i}$ fee beyond their optimal utilization. Instead, they can rely on the liquidity from the underlying layer. As such, this system is more resistant to utilization kink attacks due to a smaller $R_{high,i}/R_{low,i}$ ratio, compared to standalone pools.

5.1.2 Standalone liquidity pool

Once a protocol has matured and expanded its TVL by piggybacking off another lending pool, it may be time for the protocol owners to consider transitioning into standalone mode. This transition involves the protocol creating its own liquidity pool and transferring its assets

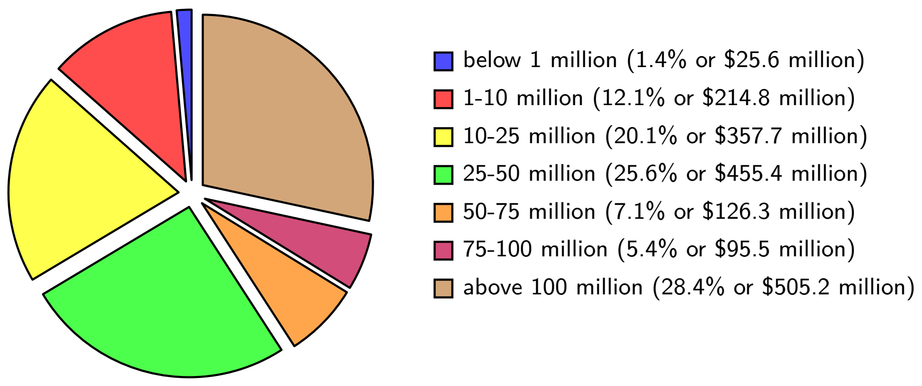


Figure 3 asset distribution beyond the top 6 protocols, totaling $1.75b.

into this new pool. It's crucial to note here that when a protocol detaches from the next one, it also severs connections with all its preceding protocols and transfers them as well. In essence, if in the chain $D_1, D_2, ..., D_i, D_{i+1}, ..., D_N, LL_N$, layer i decides to detach, it would result in two separate chains: $D_1, D_2, ..., D_i, LL_i$, and $D_i, D_{i+1}, ..., D_N, LL_N$.

Protocols should only transition to standalone mode when they have accumulated enough liquidity to fend off liquidity management attacks independently. Furthermore, during this transition, it would be advantageous for the ecosystem if the funds weren't withdrawn all at once. As these lending pools possess large liquidity pools, withdrawing all the funds abruptly could potentially trigger a spike in the underlying pools' utilization. We recommend that, at this stage, lending pools transition to a new pool by gradually vesting all the liquidity over a certain time period. For instance, a protocol could gradually withdraw all funds over the course of a day, after duly notifying the community.

6 Analyzing on-chain lending protocols

In this section, we dive into the lending pools deployed across multiple blockchain networks. Our data collection efforts aim to understand their design, TVL, and potential susceptibilities to liquidity management attacks. Our study includes two types of pools. Initially, we analyze the six most prominent lending pools in the space, and then we shift our focus to scrutinize the rest of the lending pools. Although the larger lending pools are typically secure from liquidity management attacks due to their significant liquidity base, analyzing them remains crucial as they significantly influence numerous emerging lending protocols.

According to reports [10], lending pools on the chain hold over $13.2b in TVL. Of this amount, 86.6% resides within the top six lending pools. We examine each of these influential pools, recognizing their role as templates and foundations for subsequent projects, which may adapt and develop their logic.

We also analyze smaller pools to determine their potential vulnerability to liquidity management attacks. These pools hold over $1.75b across 240 protocols on various chains, posing a tempting target for potential attackers. As shown in Figure 4 our investigation reveals that 32.5% of all 240 smaller lending pools are officially forks of Compound, while over 10% have branched off from Aave. Among the remaining 132 pools, many draw inspiration from the design choices of more established protocols, including aspects such as interest rate determination, supply, borrowing, and liquidation mechanisms. Figure 3 illustrates the distribution of funds across these protocols. When comparing the liquidity distribution of

Table 2 Data describing the six largest lending pools.

Protocol	TVL Amount	Number of Markets	Interest rate model	Liquidity Management attacks
Aave	\$5.46b	13	Aave Model	Vulnerable
JustLend	\$3.78b	1	Aave Model	Vulnerable
Compound	\$1.92b	4	Compound Model	Vulnerable
Venus	\$804.55m	1	Compound Model	Vulnerable
Morpho	\$341.38m	3	P2P/Compound Model	Possible
Radiant	\$260.09m	3	Aave Model	Vulnerable

smaller pools with the daily trading volume of Aave, which has consistently exceeded \$30 million since the start of 2023, it becomes plausible that such amount of funds is not out of reach for users in the network. Given this amount of funds, attackers could potentially execute the mentioned attacks on these pools.

Our analysis comprises a selection of noteworthy protocols, including Aave, Compound, JustLend [17], Venus [28], Morpho [13], and Radiant [25]. You can find the detailed information in Table 2. In the subsequent part of this section, we will delve into each aspect and investigate whether any of the protocols employ innovative approaches:

- **TVL:** We examine the amount of TVL each market holds and the degree of liquidity concentration which is shown by the number of markets. It's common for protocols to be deployed on multiple chains for user accessibility. Additionally, protocols often release new versions over time. While users typically prefer the latest versions, older versions can coexist and continue to serve users. For example, despite the launch of Compound V3 in August 2022, a substantial sum, exceeding \$1.32 billion, is still locked in Compound V2.
- **Supply and Borrow Mechanism:** Most lending pools utilize a similar supply and borrow mechanism, consistent with the one we outlined in our model. However, some protocols incorporate different logic, like P2P lending, and impose additional restrictions. Morpho, for instance, uses a P2P system to pair borrowers with lenders, transferring the borrower to the backup protocol, Compound, if the lender needs to withdraw their funding at any point. This mechanism makes Morpho somewhat resistant to liquidity management attacks, as borrowers borrowing from honest liquidity providers remain secure.
- **Interest Rate Model:** The interest rate model we presented in this paper generalizes those used in the mentioned protocols. Typically, smaller pools widely adopt two main models, those being Compound and Aave, due to their proven efficacy and popularity. The Compound model aligns with the model we utilized in this paper, while Aave's model, though similar, employs different variables:

$$R = \begin{cases} R'_0 + R'_{low} \times \frac{U}{kink} & if\ U \leq kink \\ R'_0 + R'_{low} + R'_{high} \times \frac{U-kink}{1-kink} & if\ U > kink \end{cases} \tag{18}$$

  Even though the formulas bear strong resemblances, they are provided to allow readers to reason with numerical examples. Aave also offers users a choice between stable and variable rates. In this paper, we presumed that protocols only offer variable rates for simplicity. Although stable rates do not alter the assumptions and results of our analysis, we direct the reader to the Aave white paper for more information on stable rates [2].
- **Attack Vulnerability:** We assess whether the pool is generally susceptible to liquidity management attacks. In each case, we assume the attacker possesses ample funds and is pursuing a specific objective. This section highlights the importance of design choices for

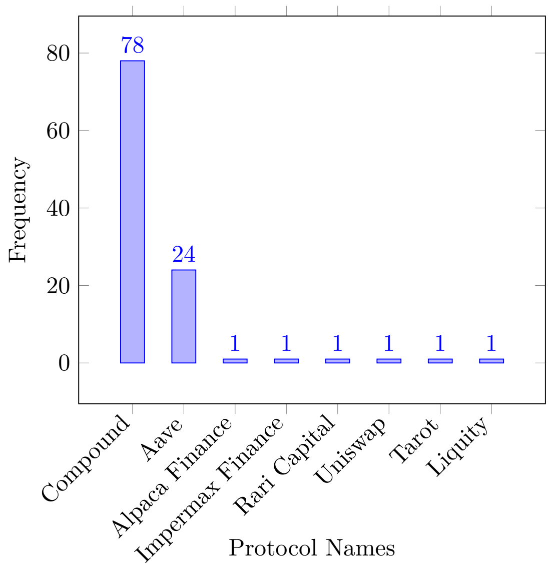


Figure 4 Frequency of protocols forked by newer projects.

new protocols adopting each of these larger protocols' designs during their initial public usage, a phase when they may have limited overall liquidity and thus be vulnerable to potential exploitation by an attacker.

7 Related work

Gudgeon et al.[15] use the term *Protocols for Loanable Funds (PLF)* to denote markets for loanable funds. Their work classifies various interest rate models utilized by leading lending protocols, including the " kinked rates" model, which is widely used by the protocols examined in our study. Bartoletti et al.[4] conceptualize the overall structure of lending pools as a state machine, analyzing different state transitions and potential threats. They introduced concepts such as over-utilization and under-utilization attacks, where attackers drive the utilization to its maximum or minimum. Sun et al.[27] explore various liquidity risks, using Aave as a case study to emphasize the significance of the issue. Hafner et al.[16] assess the degree of centralization among liquidity providers in a pool, identifying scenarios where low initial centralization could lead to liquidity shortages following substantial withdrawals. Our work extends these studies by defining liquidity management attacks and examining the motivations of a potential attacker.

8 Conclusion and future work

In this paper, we have introduced and formalized two liquidity management attacks, where an attacker with sufficient resources can exploit specific conditions within lending pools. We have demonstrated that such attacks are not only feasible but also incentivized, given the considerable amount of liquidity dispersed across numerous small liquidity pools. We further explored possible mitigation strategies and risks at the application layer that could aid upcoming lending protocols.

We additionally analyzed a specific design, wherein the design and application layer are structured as separate systems that can interact with each other. This structure enhances the flexibility of options available to liquidity pools and allows for the combination of multiple design layers that can utilize the same liquidity pool. While we scrutinized the overarching design of such systems, there remain considerable complexities to be addressed in their implementation. It is our hope that new lending pools will adopt this design and potentially establish a standard set of defensive mechanisms against liquidity management attacks.

References

1 Aave protocol website, 2023. URL: https://aave.com/.

2 Aave protocol whitepaper v1.0, 2020. URL: https://github.com/aave/aave-protocol/blob/master/docs/Aave_Protocol_Whitepaper_v1_0.pdf.

3 Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts. Cryptology ePrint Archive, Paper 2016/1007, 2016. URL: https://eprint.iacr.org/2016/1007.

4 Massimo Bartoletti, James Hsin yu Chiang, and Alberto Lluch-Lafuente. Sok: Lending pools in decentralized finance, 2020. arXiv:2012.13230.

5 Bnb bridge - rekt, 2022. URL: https://rekt.news/bnb-bridge-rekt/.

6 Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121, 2015. doi:10.1109/SP.2015.14.

7 Compound protocol website, 2023. URL: https://compound.finance/.

8 Simon Cousaert, Jiahua Xu, and Toshiko Matsui. SoK: Yield aggregators in DeFi. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, May 2022. doi:10.1109/icbc54727.2022.9805523.

9 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges, 2019. arXiv:1904.05234.

10 Defillama, 2023. URL: https://defillama.com/.

11 Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain, 2019. arXiv:1902.05164.

12 Flashbots documentation, 2023. URL: https://docs.flashbots.net/.

13 Mathis Gontier Delaunay, Quentin Garchery, Paul Frambot, Merlin Égalité, Julien Thomas, and Katia Babbar. Morpho V1 Yellow Paper. working paper or preprint, May 2023. URL: https://hal.science/hal-04087388.

14 Lewis Gudgeon, Daniel Perez, Dominik Harz, Benjamin Livshits, and Arthur Gervais. The decentralized financial crisis, 2020. arXiv:2002.08099.

15 Lewis Gudgeon, Sam M. Werner, Daniel Perez, and William J. Knottenbelt. Defi protocols for loanable funds: Interest rates, liquidity and market efficiency, 2020. arXiv:2006.13922.

16 Matthias Hafner, Romain de Luze, Nicolas Greber, Juan Beccuti, Benedetto Biondi, Gidon Katten, Michelangelo Riccobene, and Alberto Arrigoni. Defi lending platform liquidity risk: The example of folks finance: Published in the journal of the british blockchain association, April 2023. URL: https://jbba.scholasticahq.com/article/74150-defi-lending-platform-liquidity-risk-the-example-of-folks-finance.

17 Justlend dao money market protocol v1.0, December 2020. URL: https://portal.justlend.org/docs/justlend_whitepaper_en.pdf.

18 Robert Leshner and Geoffrey Hayes, February 2019. URL: https://compound.finance/documents/Compound.Whitepaper.pdf.

19 Amani Moin, Kevin Sekniqi, and Emin Gun Sirer. Sok: A classification framework for stablecoin designs. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 174–197, Cham, 2020. Springer International Publishing.

20 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, May 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

21 OpenZeppelin. Openzeppelin/openzeppelin-contracts: Openzeppelin contracts is a library for secure smart contract development. URL: https://github.com/OpenZeppelin/openzeppelin-contracts.

22 Poly network - rekt, 2021. URL: https://rekt.news/polynetwork-rekt/.

23 Kaihua Qin, Liyi Zhou, Yaroslav Afonin, Ludovico Lazzaretti, and Arthur Gervais. Cefi vs. defi – comparing centralized to decentralized finance, 2021. arXiv:2106.08157.

24 Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An empirical study of DeFi liquidations. In *Proceedings of the 21st ACM Internet Measurement Conference*. ACM, November 2021. doi:10.1145/3487552.3487811.

25 Radiant documentation, 2023. URL: https://docs.radiant.capital/radiant/.

26 Huobi Research. Global crypto industry overview and trends[2022–2023 annual report](first part), December 2022. URL: https://medium.com/huobi-research/global-crypto-industry-overview-and-trends-2022-2023-annual-report-first-part-e15372f29c.

27 Xiaotong Sun, Charalampos Stasinakis, and Georgios Sermpinis. Liquidity risks in lending protocols: Evidence from aave protocol, 2023. arXiv:2206.11973.

28 Venus protocol documentation, 2023. URL: https://docs.venus.io/docs/getstarted.

29 Anton Wahrstätter, Jens Ernstberger, Aviv Yaish, Liyi Zhou, Kaihua Qin, Taro Tsuchiya, Sebastian Steinhorst, Davor Svetinovic, Nicolas Christin, Mikolaj Barczentewicz, and Arthur Gervais. Blockchain censorship, 2023. arXiv:2305.18545.

30 Anton Wahrstätter, Liyi Zhou, Kaihua Qin, Davor Svetinovic, and Arthur Gervais. Time to bribe: Measuring block construction market, 2023. arXiv:2305.16468.

31 Sam M. Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J. Knottenbelt. Sok: Decentralized finance (defi), 2022. arXiv:2101.08778.

32 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

33 Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. SoK: Decentralized exchanges (DEX) with automated market maker (AMM) protocols. *ACM Computing Surveys*, 55(11):1–50, February 2023. doi:10.1145/3570639.

34 Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges, 2020. arXiv:2009.14021.

35 Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. Sok: Decentralized finance (defi) attacks, 2023. arXiv:2208.13035.

Analysis of CryptoNote Transaction Graphs Using the Dulmage-Mendelsohn Decomposition

Saravanan Vijayakumaran
Department of Electrical Engineering, Indian Institute of Technology Bombay, Mumbai, India

Abstract

CryptoNote blockchains like Monero represent the largest public deployments of linkable ring signatures. Beginning with the work of Kumar et al. (ESORICS 2017) and Möser et al. (PoPETs 2018), several techniques have been proposed to trace CryptoNote transactions, i.e. identify the actual signing key, by using the transaction history. Yu et al. (FC 2019) introduced the closed set attack for undeniable traceability and proved that it is optimal by showing that it has the same performance as the brute-force attack. However, they could only implement an approximation of the closed set attack due to its exponential time complexity. In this paper, we show that the Dulmage-Mendelsohn (DM) decomposition of bipartite graphs gives a polynomial-time implementation of the closed set attack. Our contribution includes open source implementations of the DM decomposition and the clustering algorithm (the approximation to the closed set attack proposed by Yu et al). Using these implementations, we evaluate the empirical performance of these methods on the Monero dataset in two ways – firstly using data only from the main Monero chain and secondly using data from four hard forks of Monero in addition to the main Monero chain. We have released the scripts used to perform the empirical analysis along with step-by-step instructions.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Cryptocurrency, CryptoNote, Monero, Traceability

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.28

Supplementary Material *Software*: `https://github.com/avras/cryptonote-analysis`
archived at `swh:1:dir:1cdef0560817a968586847a625465093dec593c4`
Other (Documentation): `https://www.respectedsir.com/cna`

Acknowledgements We thank Justin Ehrenhofer for sharing the blockchain databases of the (no longer operational) Monero Original, MoneroV, Monero v7, and Monero v9 forks [12]. We also thank him for his feedback on an earlier version of this paper, which helped improve the presentation of the empirical results. We thank Zuoxia Yu for sharing the full version of their FC 2019 paper. Finally, we thank the anonymous reviewers of PoPETs 2022 (where an earlier version of this paper was eventually rejected) and of the current conference for their comments. We prepared the instructions for reproducing our empirical results on the suggestion of a PoPETs 2022 reviewer.

1 Introduction

1.1 CryptoNote Transactions

Coins in CryptoNote blockchains are associated with stealth addresses, which are also called one-time addresses or transaction outputs [21]. We will use the term output for brevity. Each output is uniquely identified by a public key, which is a point on an elliptic curve. To spend from an output, the spender needs to know the corresponding secret key.

In a transaction, the spender creates a *ring* of outputs which is a set containing the output being spent and some other outputs sampled from the CryptoNote blockchain (these are called decoy outputs or *mixins*). The spender generates a linkable ring signature [9] over the ring of outputs using the secret key of the output being spent. This signature only reveals that the signer knows the secret key corresponding to one of the ring outputs, without revealing the identity of the actual output being spent.

To prevent double spending from an output, the linkable ring signature reveals the *key image* of the output being spent. The key image of an output is a collision-resistant one-way function of the secret key. For example, in Monero the public key associated with an output is given by $P = xG$ where G is the base point of an elliptic curve group and x is the secret key. The key image I of the output associated with P is given by $xH_{\mathrm{p}}(P)$, where $H_{\mathrm{p}}(\cdot)$ is a cryptographic hash function that maps its input to a point on the elliptic curve. In the rest of this paper, when we speak of *the key image of an output* we mean the result of the one-way function applied to the secret key associated with the output.

If the owner of the output corresponding to P tries to spend the coins associated with it more than once, then the key image I would appear again in the second transaction, identifying it as a double spending transaction. Such transactions will be rejected by miners, as blocks including them would be considered invalid by the network. In this sense, the key image acts as a *nullifier* of an output, ensuring that it is spent only once.

1.2 CryptoNote Transaction Graphs

Consider a CryptoNote transaction which spends from two existing outputs and creates three new outputs as illustrated in Figure 1. The new outputs are denoted by T_1, T_2, T_3. The transaction has two rings of outputs of size five each, $(P_1, P_2, \ldots, P_5)$ and $(Q_1, Q_2, \ldots, Q_5)$. Exactly one output from each ring is being spent in the transaction. The key images I_1 and I_2 of the outputs being spent are revealed in the transaction. Note that the two rings can have common outputs.

For the purpose of illustration, suppose that the two rings have two outputs in common. Let $Q_1 = P_4$ and $Q_2 = P_5$. The relationship between the ring outputs and the key images in this transaction can be represented by the bipartite graph in Figure 2. The union of the two ring output sets forms one vertex class and the two key images form the other vertex class. An edge between an output and a key image indicates that the latter could be the true key image of that output. Note that the new outputs T_1, T_2, T_3 play no role in the construction of the bipartite graph. We will refer to such output/key image bipartite graphs as *transaction graphs*.

As each key image must have been generated from a unique output, any pair of edges (P_i, I_1) and (Q_j, I_2) such that $P_i \neq Q_j$ is a plausible candidate for the true relationship between the outputs and key images. A *matching* on a graph is a subset of the edges such

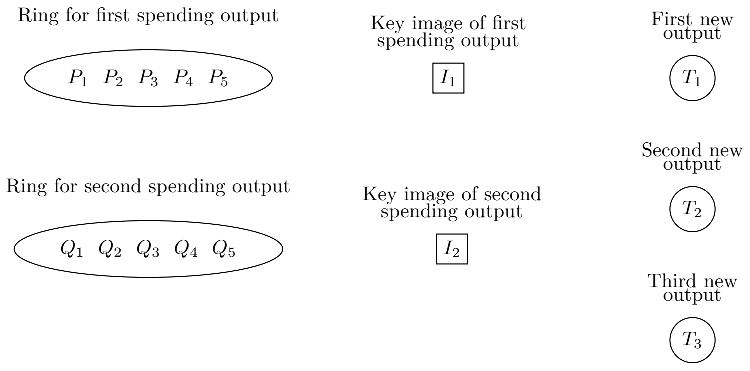

Ring for first spending output
P_1 P_2 P_3 P_4 P_5
Key image of first spending output
I_1
First new output
T_1
Ring for second spending output
Q_1 Q_2 Q_3 Q_4 Q_5
Key image of second spending output
I_2
Second new output
T_2
Third new output
T_3

Figure 1 A CryptoNote transaction with two inputs and three outputs.

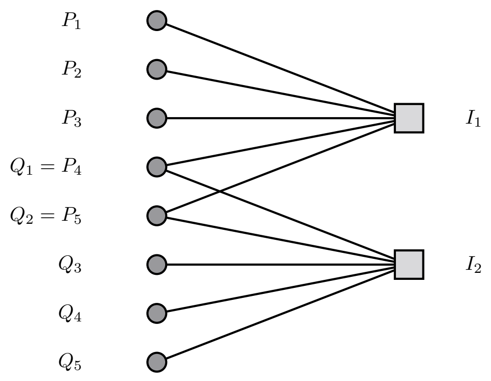


Figure 2 Transaction graph corresponding to the transaction in Figure 1.

that no two edges in the subset share a vertex (see Section 5.2 for a precise definition). The pair of edges (P_i, I_1) and (Q_j, I_2) with $P_i \neq Q_j$ is a matching on the graph in Figure 2. In fact, it is a matching of maximum size as any three edges in this graph would have two which meet in either I_1 or I_2.

Let us now consider a similar bipartite graph induced by the set of all transactions which have appeared up to the block having height h. The key image vertex class $\mathcal{K}_h$ in this graph is the set of all key images which have appeared on the blockchain up to block height h. The output vertex class $\mathcal{O}_h$ is the set of all outputs which have *appeared in at least one transaction ring* in the blocks up to height h. Note that $\mathcal{O}_h$ is *not* the set of all outputs which have appeared on the blockchain in blocks up to height h. We represent the edge set of the transaction graph induced by the CryptoNote transaction rings as a subset E of $\mathcal{O}_h \times \mathcal{K}_h$. For $P \in \mathcal{O}_h$ and $I \in \mathcal{K}_h$, the edge (P, I) belongs to E if the output P appeared in the transaction ring used to create I (via the linkable ring signature).

Since each key image $I \in \mathcal{K}_h$ is generated from a unique output $P \in \mathcal{O}_h$, we have $|\mathcal{K}_h| \leq |\mathcal{O}_h|$. In a bipartite graph with vertex classes of cardinality m and n, the size of a maximum matching can be at most $\min(m, n)$. Since the edges corresponding to the *true association* between outputs and key images form a matching of size $|\mathcal{K}_h|$, the induced bipartite graph always has a maximum matching. In fact, we have the following principle which has been discussed by Monero Research Lab researchers [6] and others [25, 26].

▶ **Observation 1.** *Any maximum matching on a CryptoNote transaction graph is a plausible candidate for the ground truth, i.e. the true association between outputs and key images.*

1.3 Tracing CryptoNote Transactions

While a single linkable ring signature over a ring of public keys guarantees signer anonymity against computationally bounded adversaries [9], CryptoNote blockchains typically have signatures created using overlapping rings which can reveal the identity of the signing key. In this context, the *signing key* is the public key corresponding to the secret key which was used to generate the linkable ring signature.

▶ **Definition 2.** *A CryptoNote transaction ring is said to be* ***traceable*** *if the true signing key is correctly identified.*

To illustrate how CryptoNote transaction rings can be traced, consider three CryptoNote transaction rings having ring members $\{P_1\}$, $\{P_1, P_2\}$, and $\{P_1, P_2, P_3, P_4\}$ respectively. Let I_1, I_2, I_3 be the distinct key images created from these three transaction rings. The corresponding CryptoNote transaction graph is shown in Figure 3.

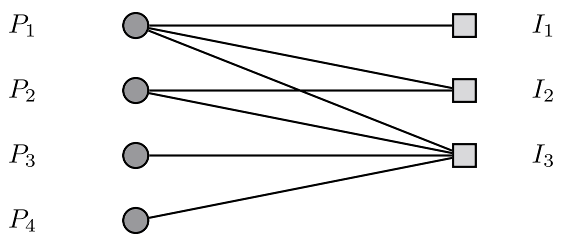


Figure 3 Transaction graph corresponding to three CryptoNote transactions.

(a) The first transaction ring has only one member and is therefore trivially traceable. The key P_1 must be the signing key. Such transactions are called *zero-mixin transactions*.
(b) In isolation, either P_1 or P_2 could have been the signing key in the second transaction ring. But once P_1 has been identified as the signing key in the first transaction, we know that I_1 is the key image corresponding to P_1. Since the signing key in the second transaction ring has key image $I_2 \neq I_1$, it must be equal to P_2.
(c) Similarly, we can eliminate P_1 and P_2 from the set of possible signing keys of the third transaction ring. Any one of the remaining two ring members, P_3 and P_4, can the true signing key of this ring. There is not enough information to conclusively identify one of them as the true signing key.

In this example, the first two transaction rings are traceable and the third one is not. But the effective mixin size of the third transaction ring was reduced from 3 to 1. So the presence of traceable rings can affect the privacy of other untraceable rings.

1.4 Paper Organization

We present related work in Section 2 followed by a summary of our contributions in Section 3. The closed set attack and the clustering algorithm for approximating it are described in Section 4. In Section 5, we describe the Dulmage-Mendelsohn (DM) decomposition. In Section 6, we show that the DM decomposition finds all possible closed sets in a CryptoNote transaction graph. In Section 7, we describe the empirical results obtained by applying the DM decomposition to the Monero transaction graph with and without information from hard forks. Section 8 concludes the paper.

2 Related Work

The first traceability analyses on CryptoNote blockchains were performed by Kumar et al. [8] and Möser et al. [17]. They both studied the Monero blockchain history and found that zero-mixin transactions have a cascade effect of rendering other transactions traceable. This technique for achieving undeniable traceability is called the *cascade attack*.[1] They also considered heuristics for tracing transactions like the guess-newest heuristic and the output merging heuristic. But these methods only achieve plausible traceability and can lead to false positives.

In our survey of related work, we restrict our attention to methods for undeniable traceability. Readers interested in methods for plausible traceability can refer to [8, 17, 23, 18, 1]. A line of work describing the design of better ring samplers and sustainable ring-based anonymous systems can be found in [20, 5, 2].

[1] The second transaction ring in the example of Section 1.3 was traced using the cascade attack.

The cascade attack proceeds in an iterative manner. First, it marks the outputs in zero-mixin transactions as spent. Then it marks these outputs as mixins in other (non-zero-mixin) transactions. If all outputs except one in a transaction ring are marked as mixins, then the remaining output is identified as the output being spent (and the transaction becomes undeniably traceable). The outputs which have been newly marked as spent in a ring are marked as mixins in other rings. The process continues until no new outputs can be marked as spent.

The initial implementation of Monero did not hide the transaction amounts. In January 2017, Monero introduced a new transaction type called *ring confidential transaction (RingCT)*, where transaction output amounts are hidden in Pedersen commitments. RingCT became mandatory in September 2017 [15].

While the cascade attack was able to trace a significant percentage of non-RingCT transactions, RingCT transactions remain immune to it. In our empirical evaluation, we found that the cascade attack could not trace any RingCT transactions up to block height 2,530,000. This was primarily because RingCT transactions did not allow zero-mixin rings.

Wijaya et al. [22] observed that a zero-mixin effect could be created in RingCT transactions by spending n times from a ring of size n. The n outputs in the ring can then be marked as mixins in other transaction rings. They name this type of spending behavior the *ring attack*. As a proof of concept, they created five outputs in Monero block 1,468,425 and then spent all of them using the other four as mixins in five transaction rings in block 1,468,439. This behavior does not arise naturally due to the mixin sampling strategy in Monero. Up to Monero block 2,530,000 (January 4, 2022), the ring of size 5 created by Wijaya et al. is the only RingCT ring which exhibits this behavior.

Yu et al. [26] defined a *closed set* to be a set of n outputs which can be represented as a union of n transaction rings. As each transaction ring must spend a unique output, the outputs in a closed set can be marked spent. They proved that the *closed set attack* (which finds all closed sets) is optimal by showing that its output is equivalent to the output of a brute-force attack. However, they observed that the naive method of finding closed sets by testing all subsets of the outputs has exponential time complexity.

As a workaround, they proposed an approximate algorithm to identify closed sets called the *clustering algorithm*. After executing the cascade attack [8, 17] on the transaction rings, the clustering algorithm attempts to find closed sets by combining transaction rings which are close to each other (see Section 4 for a more detailed description). They proved that the clustering algorithm can identify all closed sets up to size 5. While the performance of closed set attack is better than the cascade attack, they reported that no RingCT transactions were traced by their algorithm.

Several projects have forked the Monero blockchain resulting in multiple blockchains with large numbers of common outputs. When a common output is spent in two different forks, the same key image appears in both spending transactions. The real output is then contained in the intersection of the transaction rings of such transactions. Wijaya et al. [24] and Hinteregger et al. [7] used repeated key images which appeared in Monero and two hard forks, Monero Original [14] and MoneroV [16], to trace transactions in all three chains. While their methods were the first ones which successfully traced RingCT transactions in Monero, they reported that the overall impact of their techniques was small. Only a small percentage of the total set of transactions were rendered traceable. Wijaya et al. [24] also discuss strategies for mitigating the loss of anonymity due to key reuse in hard forks.

The Monero reference implementation includes a tool for identifying spent outputs using the techniques described above [13]. It implements the cascade attack, finds transactions which cause the ring attack characterized by Wijaya et al. [22], attempts to identify closed sets, and performs the cross-chain analysis proposed by Wijaya et al. [24] and Hinteregger et al. [7]. It is included in every Monero release as an executable with the name monero-blockchain-mark-spent-outputs. It is informally called the "blackball tool" in the Monero community as the set of spent outputs represent a blacklist which should be avoided when sampling mixins. To perform cross-chain analysis, the tool takes the LMDB database files of all chains as input.

We noticed two issues with the Monero blackball tool with regard to cross-chain analysis. Firstly, the tool is not able to read the LMDB database of MoneroV [16] due to a discrepancy in the transaction formats in the main Monero code and the MoneroV code. This discrepancy did not affect our analysis as we used the JSON-RPC interface of the MoneroV client [16] to extract the transaction data. Secondly, and more seriously, the tool only uses an integer index to uniquely identify an output across chains and not the output public key.

Outputs in a single Monero chain are partitioned by their amounts (with RingCT outputs having dummy amount zero) and are assigned increasing indices in the order of their appearance on the chain. This means that outputs which appear in two different chains after a fork can have the same index even though they have different public keys. Consequently, the cross-chain analysis performed by the blackball tool has errors. To be fair, the tool outputs error messages during its execution when it encounters disjoint transaction rings for the same key image. The presence of code for generating these error messages suggests that the blackball tool developers are aware of this issue. To avoid such errors, we used the public keys of the outputs as their unique identifier in our cross-chain analysis.

3 Our Contributions

Our contributions are as follows.

1. Our main contribution is to show that the Dulmage-Mendelsohn (DM) decomposition of bipartite graphs gives an efficient implementation of the closed set attack, which is the optimal method for undeniable traceability in CryptoNote blockchains. Computing the DM decomposition involves finding a maximum matching, a depth-first search from all unmatched vertices, and a computation of strongly connected components, all on the CryptoNote transaction graph. All three algorithms have polynomial time complexity in the number of nodes and edges of the graph.
2. We implemented the DM decomposition, the cascade attack, and the closed set attack in Rust. The code is available at `https://github.com/avras/cryptonote-analysis` under an MIT license. While an open source implementation of the DM decomposition already existed in CSparse [3], it was much slower than the proprietary implementation in Matlab [11]. Our implementation of the DM decomposition has performance comparable to the Matlab implementation.
3. We compute the empirical performance of the DM decomposition method on Monero and show that it outperforms the clustering algorithm approximation to the closed set attack proposed by Yu et al. [26].
4. While previous traceability attacks have been effective against non-RingCT transactions in Monero, RingCT transactions have been mostly immune. Only cross-chain analysis which uses information from hard forks has been able to trace Monero RingCT transactions [7]. We compute the empirical performance of the DM decomposition method using information from four different hard forks: Monero Original, MoneroV, Monero v7, and Monero v9. Our results show that, even with hard fork information, Monero RingCT transactions are mostly immune to undeniable traceability via the DM decomposition method.

5. We released the scripts used to generate our empirical results in the code repository at `https://github.com/avras/cryptonote-analysis`. We prepared detailed instructions on how to reproduce our results and made them available at `https://www.respectedsir.com/cna`.

4 The Closed Set Attack and Clustering Algorithm

In a CryptoNote blockchain, let $R_i = \{P_{i,1}, P_{i,2}, \ldots, P_{i,n_i}\}$ be the ring of public keys in the ith transaction. Let $\mathcal{R}_h = \{R_1, R_2, \ldots, R_n\}$ be the multiset[2] of all transaction rings which have appeared on the blockchain up to height h. Since each ring has a unique key image associated with it, the set of key images $\mathcal{K}_h$ has size n.

As per the notation introduced in Section 1.2, the set of outputs that have appeared in at least one transaction ring is given by $\mathcal{O}_h = \cup_{i=1}^{n} R_i$. Let $m = |\mathcal{O}_h|$. As the number of key images cannot exceed the number of public keys, we have $m \geq n$.

4.1 Brute-Force Attack

Yu et al. [26] proposed the closed set attack and argued that it is optimal because of having identical traceability performance to the *brute-force attack*. We will use the notion of a *system of distinct representatives* [10] to describe the brute-force attack.

▶ **Definition 3.** *Let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be a multiset of subsets of a finite set S. A set of n distinct elements $\{s_1, s_2, \ldots, s_n\}$ which satisfies $s_i \in S_i$ is called a* ***system of distinct representatives (SDR)*** *for $\mathcal{S}$.*

In the context of CryptoNote blockchains, an SDR $\{P_1, P_2, \ldots, P_n\}$ for the multiset $\mathcal{R}_h = \{R_1, R_2, \ldots, R_n\}$ corresponds to a possible candidate for the sequence of signing keys for the rings in $\mathcal{R}_h$. This is because for each i the key P_i belongs to R_i and all the P_i's are distinct.

The brute-force attack for tracing CryptoNote transactions is shown in Algorithm 1. This attack prunes the rings in $\mathcal{R}_h$ to generate the multiset $\mathcal{R}'_h = \{R'_1, R'_2, \ldots, R'_n\}$ where each R'_i is a subset of R_i. It includes only those keys in R'_i which are potential signing keys for the ring R_i.

The time complexity of the brute-force attack is $O\left(\prod_{i=1}^{n} |R_i|\right)$ as each P_i can be independently chosen in $|R_i|$ ways. If the number of rings R_i in $\mathcal{R}_h$ with at least two keys is n_2, then $\prod_{i=1}^{n} |R_i| \geq 2^{n_2}$. Thus the brute-force attack becomes infeasible as the number of rings with at least one mixin increases.

4.2 Closed Set Attack

Yu et al. [26] define a closed set as follows.

▶ **Definition 4.** *Let $\mathcal{R}_h = \{R_1, R_2, \ldots, R_n\}$ be a multiset of CryptoNote transaction rings and $\mathcal{O}_h = \cup_{i=1}^{n} R_i$. A subset C of $\mathcal{O}_h$ having cardinality k is called a* ***closed set*** *of $\mathcal{R}_h$ if there exist k transaction rings $R_{i_1}, R_{i_2}, \ldots, R_{i_k}$ in $\mathcal{R}_h$ such that $C = \cup_{j=1}^{k} R_{i_j}$.*

[2] In a multiset, elements can occur more than once.

Algorithm 1 The brute-force attack for tracing CryptoNote transactions.

```
Input  : A multiset of transaction rings $\mathcal{R}_h = \{R_1, R_2, \ldots, R_n\}$
Output : A multiset of pruned transaction rings $\mathcal{R}'_h = \{R'_1, R'_2, \ldots, R'_n\}$ where
         $\emptyset \neq R'_i \subseteq R_i$ for all $i \in \{1, 2, \ldots, n\}$
// Initialize the rings in $\mathcal{R}'_h$ to the empty set;
for $i \leftarrow 1$ to $n$ do
    $R'_i \leftarrow \emptyset$
end
// Iterate over elements of $R_1 \times R_2 \times \cdots \times R_n$ to find SDRs;
foreach $(P_1, P_2, \ldots, P_n)$ in $R_1 \times R_2 \times \cdots \times R_n$ do
    if $\{P_1, P_2, \ldots, P_n\}$ is an SDR then
        for $i \leftarrow 1$ to $n$ do
            // Add $P_i$ to $R'_i$;
            $R'_i \leftarrow R'_i \cup \{P_i\}$
        end
    end
end
```

▶ **Example 5.** To illustrate the definition and significance of a closed set, suppose that $\mathcal{R}_h$ contains four transaction rings of the following form.

$$R_1 = \{P_1, P_2, P_3\},$$
$$R_2 = \{P_2, P_3\},$$
$$R_3 = \{P_1, P_3\},$$
$$R_4 = \{P_1, P_2, P_3, P_4\}.$$

Exactly one public key from each ring is used to generate the linkable ring signature. If a public key was used twice, the key image would repeat and the later of the two transactions would be rejected by the miners. Since the union of the first three rings $R_1 \cup R_2 \cup R_3 = \{P_1, P_2, P_3\}$ has cardinality 3, the set $\{P_1, P_2, P_3\}$ is a closed set. Each of the keys P_1, P_2, P_3 must be the signing key in exactly one of the first three rings. This implies that none of them can be the signing key in the fourth ring R_4. So we deduce that P_4 must be the signing key of ring R_4, rendering the latter traceable. ⌟

The closed set attack for tracing CryptoNote transactions is shown in Algorithm 2. Yu et al. [26] proved that it is is optimal for undeniable traceability by showing that its output is identical to the output of the brute-force attack. We rephrase their Theorem 1 in our notation as follows.

▶ **Theorem 6** (Yu et al. [26]). *Given a multiset $\mathcal{R}_h = \{R_1, R_2, \ldots, R_n\}$ of CryptoNote transaction rings, let the multiset $\mathcal{R}_h^{closed}$ be the output of the closed set attack and the multiset $\mathcal{R}_h^{brute}$ be the output of the brute-force attack. Then $\mathcal{R}_h^{closed} = \mathcal{R}_h^{brute}$.*

The running time of the closed set attack is dominated by the time required to find all the closed sets in $\mathcal{R}'_h$. Note that $\mathcal{R}'_h$ is initially equal to $\mathcal{R}_h$. After a closed set is found, some of its rings may be pruned. This may cause new closed sets to become available. Hence the search needs to be performed again.

Algorithm 2 The closed set attack for tracing CryptoNote transactions.

```
Input  : A multiset of transaction rings $\mathcal{R}_h = \{R_1, R_2, \dots, R_n\}$
Output : A multiset of pruned transaction rings $\mathcal{R}'_h = \{R'_1, R'_2, \dots, R'_n\}$ where
         $\emptyset \neq R'_i \subseteq R_i$ for all $i \in \{1, 2, \dots, n\}$
// Initialize the rings in $\mathcal{R}'_h$ to the rings in $\mathcal{R}_h$;
for $i \leftarrow 1$ to $n$ do
    $R'_i \leftarrow R_i$
end
// Iterate over all the closed sets;
foreach closed set $C = \cup_{j=1}^{k} R'_{i_j}$ do
    for $i \leftarrow 1$ to $n$ do
        // Check that the ring does not generate the closed set;
        if $i \notin \{i_1, i_2, \dots, i_k\}$ then
            // Remove elements of $C$ from $R'_i$;
            $R'_i \leftarrow R'_i \cap C^c$
        end
    end
end
```

The naive algorithm [26, Appendix A] for finding closed sets by considering all subsets of $\mathcal{R}'_h$ becomes infeasible as the size of $|\mathcal{R}_h|$ increases. Instead, Yu et al. [26] proposed the clustering algorithm as an approximation to the naive algorithm.

4.3 Clustering Algorithm

▶ **Definition 7.** *A subset of $\mathcal{R}_h$ is called a **cluster**.*

A cluster consists of a set of rings from $\mathcal{R}_h$. The distance of a ring R from a cluster is defined as follows.

▶ **Definition 8.** *Let $\mathcal{C} = \{R_{i_1}, R_{i_2}, \dots\}$ be a cluster and let $pk(\mathcal{C}) = \cup_{R' \in \mathcal{C}} R'$ be the set of keys in it. The **distance** of a ring $R \in \mathcal{R}_h$ from a cluster $\mathcal{C}$ is defined as*

$$d(R, \mathcal{C}) = |R| - |pk(\mathcal{C}) \cap R|.$$

▶ **Example 9.** Consider the rings $R_1, \dots, R_4$ from Example 5. Suppose we consider the cluster $\mathcal{C} = \{R_1, R_2\}$. Then we have $pk(\mathcal{C}) = \{P_1, P_2, P_3\}$, $d(R_3, \mathcal{C}) = 0$, and $d(R_4, \mathcal{C}) = 1$. ⌟

Yu et al. [26] use a cluster formation algorithm as a subroutine in their clustering algorithm. This algorithm forms a cluster by starting from any transaction ring in $\mathcal{R}_h$ and adding other rings which are at a distance of at most 1 from the running cluster (see Algorithm 3).

The clustering algorithm for finding closed sets is shown in Algorithm 4 where `Cascade-Attack` is a procedure that implements the cascade attack of [8, 17]. `CascadeAttack` takes a multiset of transactions rings as input and outputs a pruned multiset after removing keys from each ring that have been identified as mixins by the cascade attack.

The clustering algorithm may fail to find certain closed sets because a ring that is needed to form a closed set may be at a distance of 2 or more from the current cluster in Algorithm 3. We observed this in our empirical analysis of the Monero transaction graph where the clustering algorithm failed to find some closed sets that were found by the DM decomposition.

Algorithm 3 ClusterForm: An algorithm for constructing a cluster.

```
Input  : A multiset of transaction rings $\mathcal{R}_h = \{R_1, R_2, \ldots, R_n\}$ and a specific ring
         $R \in \mathcal{R}_h$
Output : A cluster $\mathcal{C}$ containing $R$
// Initialize $\mathcal{C}$ to the set containing $R$;
$\mathcal{C} \leftarrow \{R\}$
// Iterate over all the rings in $\mathcal{R}_h$ not equal to $R$;
foreach $R' \in \mathcal{R}_h \setminus R$ do
    // Check if $R'$ is within a distance of 1 from $\mathcal{C}$;
    if $d(R', \mathcal{C}) \leq 1$ then
        // Add $R'$ to $\mathcal{C}$;
        $\mathcal{C} \leftarrow \mathcal{C} \cup R'$
    end
end
```

5 The Dulmage-Mendelsohn Decomposition

Consistent with notation used by Dulmage and Mendelsohn [4], we define an undirected bipartite graph K as a triple (S, T, E) where S and T are non-empty sets representing vertex classes and $E \subseteq S \times T$ represents the edge set. So an edge in K is given by an ordered pair (s, t) where $s \in S$ and $t \in T$. The ordering of the vertices in the edge (s, t) is simply a consequence of putting S before T in the triple (S, T, E), and does not imply directivity. We say that an edge (s, t) belongs to the graph K, written as $(s, t) \in K$, to mean that $(s, t) \in E$. We only consider bipartite graphs K where both S and T are finite sets.

5.1 Minimum Covers of Bipartite Graphs

▶ **Definition 10.** *Let $K = (S, T, E)$ be a bipartite graph. Let A and B be subsets of S and T respectively. A pair of such sets (A, B) is called a* ***vertex cover*** *for a bipartite graph K if for each edge $(s, t) \in K$, either $s \in A$ or $t \in B$ (both conditions can also hold).*

▶ **Definition 11.** *The* ***size*** *of a vertex cover (A, B) is defined as $|A| + |B|$ where $|X|$ denotes the cardinality of a set X.*

Since S and T are assumed to be finite sets, every vertex cover of K will have a finite size.

▶ **Definition 12.** *The* ***cover number*** *of a bipartite graph K is the minimum of $|A| + |B|$ over all vertex covers (A, B) of K.*

▶ **Definition 13.** *A vertex cover (A, B) of a bipartite graph K whose size equals the cover number of K is called a* ***minimum cover****.*

The following two results were proved by Dulmage and Mendelsohn [4].

▶ **Lemma 14.** *If (A_1, B_1) and (A_2, B_2) are minimum covers of a bipartite graph K having finite cover number, then $(A_1 \cap A_2, B_1 \cup B_2)$ and $(A_1 \cup A_2, B_1 \cap B_2)$ are both minimum covers of K.*

▶ **Lemma 15.** *Let (A_1, B_1) and (A_2, B_2) be minimum covers of a bipartite graph K having finite cover number. If $A_1 \subseteq A_2$, then $B_1 \supseteq B_2$.*

Algorithm 4 The clustering algorithm.

```
Input   : A multiset of transaction rings $\mathcal{R}_h = \{R_1, R_2, \dots, R_n\}$
Output : A multiset of pruned transaction rings $\mathcal{R}'_h = \{R'_1, R'_2, \dots, R'_n\}$ where
          $\emptyset \neq R'_i \subseteq R_i$ for all $i \in \{1, 2, \dots, n\}$
// Run the cascade attack on $\mathcal{R}_h$;
$\mathcal{R}'_h \leftarrow$ CascadeAttack($\mathcal{R}_h$);
// Set flag to true;
flag $\leftarrow$ true;
while flag is true do
    flag $\leftarrow$ false;
    // Iterate over all the rings in $\mathcal{R}'_h$;
    foreach $R' \in \mathcal{R}'_h$ do
        // Form a cluster starting from $R'$ using Algorithm 3;
        $\mathcal{C}' = \{R'_{i_1}, R'_{i_2}, \dots, R'_{i_k}\} \leftarrow$ ClusterForm($R'$);
        if $C = \cup_{j=1}^{k} R'_{i_j}$ is a closed set then
            for $i \leftarrow 1$ to $n$ do
                // Check that the ring does not generate the closed set $C$;
                if $i \notin \{i_1, i_2, \dots, i_k\}$ then
                    // Check if $C$ and $R'_i$ have elements in common;
                    if $C \cap R'_i \neq \emptyset$ then
                        // Remove elements of $C$ from $R'_i$;
                        $R'_i \leftarrow R'_i \cap C^c$;
                        // Set the flag to indicate modification of transaction rings;
                        flag $\leftarrow$ true;
                    end
                end
            end
            if flag is true then
                // Run the cascade attack on $\mathcal{R}'_h$;
                $\mathcal{R}'_h \leftarrow$ CascadeAttack($\mathcal{R}'_h$);
                // Run the cascade attack on the cluster $\mathcal{C}'$;
                $\mathcal{C}'' = \{R''_{i_1}, R''_{i_2}, \dots, R''_{i_k}\} \leftarrow$ CascadeAttack($\mathcal{R}'_h$);
                // Replace the rings in $\mathcal{R}'_h$ with intra-cluster cascade attack results;
                for $j \leftarrow 1$ to $k$ do
                    $R'_{i_j} \leftarrow R''_{i_j}$;
                end
            end
        end
    end
end
```

Setting $A_1 = A_2$ in the above lemma gives us the following corollary.

▶ **Corollary 16.** *If (A, B_1) and (A, B_2) are both minimum covers of a bipartite graph K having finite cover number, then $B_1 = B_2$.*

For a bipartite graph K, let $\mathcal{C}$ be the set of all minimum covers. Let us define the following sets obtained by taking intersections and unions of the components of the minimum covers.

$$A_* = \bigcap_{(A,B)\in\mathcal{C}} A, \qquad A^* = \bigcup_{(A,B)\in\mathcal{C}} A, \tag{1}$$

$$B_* = \bigcap_{(A,B)\in\mathcal{C}} B, \qquad B^* = \bigcup_{(A,B)\in\mathcal{C}} B. \tag{2}$$

By Lemma 14, if K has a finite cover number then the pairs (A_*, B^*) and (A^*, B_*) are both minimum covers of K.

▶ **Example 17.** Reconsider the bipartite graph shown in Figure 3 with vertex classes $S = \{P_1, P_2, P_3, P_4\}$ and $T = \{I_1, I_2, I_3\}$. Since $(\emptyset, T)$ is a minimum cover the graph, $A_* = \emptyset$ and $B^* = T$. As $(\{P_1\}, \{I_2, I_3\})$ and $(\{P_1, P_2\}, \{I_3\})$ are the only other minimum covers, $A^* = \{P_1, P_2\}$ and $B_* = \{I_3\}$.

5.2 Maximum Matchings on Bipartite Graphs

In a graph, we say that edges (s, t) and (s', t') share a vertex if either $s = s'$ or $t = t'$.

▶ **Definition 18.** *A* ***matching*** *on a bipartite graph $K = (S, T, E)$ is a subset M of the edge set E such that no two edges in M share a vertex. The cardinality $|M|$ is called the* ***order*** *of the matching M.*

▶ **Definition 19.** *A* ***maximum matching*** *on a bipartite graph K is a matching on K of maximum order.*

The following definition classifies edges according to their membership in maximum matchings on K.

▶ **Definition 20.** *An edge (s, t) of a bipartite graph K is said to be* ***admissible*** *if there exists a maximum matching M on K such that $(s, t) \in M$. An edge which is not admissible is said to be* ***inadmissible****.*

The following result by König [10] says that maximum matchings have the same size as minimum covers in bipartite graphs. We will need it in the proof of Theorem 26.

▶ **Proposition 21.** *The cover number of a finite bipartite graph equals the order of maximum matchings on the graph.*

5.3 Definition of the DM Decomposition

With the above definitions in place, we are ready to describe the DM decomposition.

▶ **Definition 22.** *Let $K = (S, T, E)$ be a bipartite graph having a finite cover number. The* ***Dulmage-Mendelsohn decomposition*** *of K is a partition of $S \times T$ into three disjoint sets R_1, R_2, R_3 which satisfy the following properties:*

1. *The set of admissible edges in K equals $E \cap R_1$.*
2. *The set of inadmissible edges in K equals $E \cap R_2$.*
3. *$E \cap R_3 = \emptyset$.*

The fine structure of the sets R_1, R_2, R_3 depends on the minimum covers of K. Let us consider two cases.

5.3.1 Case 1: $A_* = A^*$

If $A_* = A^*$, then the graph K has only one minimum cover given by $(A_*, B^*) = (A^*, B_*)$. In this case, the following result holds.

▶ **Proposition 23.** *Let $K = (S, T, E)$ be a bipartite graph having a finite cover number. If K has only one minimum cover given by (A_*, B^*), then the Dulmage-Mendelsohn decomposition of K is the partition of $S \times T$ into the sets R_1, R_2, R_3 given by*

$$\begin{aligned} R_1 &= (A_* \times (B^*)^c) \bigcup ((A_*)^c \times B^*)\,, \\ R_2 &= A_* \times B^*, \\ R_3 &= (A_*)^c \times (B^*)^c. \end{aligned} \tag{3}$$

5.3.2 Case 2: $A_* \neq A^*$

Now suppose $A_* \neq A^*$. By definition, $A_* \subseteq A^*$. So A_* must be a proper subset of A^*. Then there exists at least one non-empty set $X \subset S$ such that $A_* \cap X = \emptyset$ and $(A_* \cup X, Y)$ is a minimum cover of K for some $Y \subset T$. The existence of such a set follows from the fact $A^* \setminus A_*$ is a candidate for X. Let S_1 be a set of smallest cardinality among all candidates for X. There may be many possibilities for S_1, all having the same smallest cardinality. We can pick any one of them.

Let (A_1, B_1) be a minimum cover with $A_1 = A_* \cup S_1$. By Corollary 16, B_1 is uniquely determined by A_1. As $A_* \subseteq A_1$, Theorem 15 tells us that $B_1 \subseteq B^*$. As all minimum covers of K have the same size, we have $|A_*|+|B^*| = |A_1|+|B_1|$. Since $|A_1| > |A_*|$, we have $|B_1| < |B^*|$. Thus B_1 is a proper subset of B^*. Let $T_1 = B^* \setminus B_1$. Since $|A_1| - |A_*| = |B^*| - |B_1|$, we have $|S_1| = |T_1|$.

If $A_1 = A^*$, the process stops. Otherwise, there exists at least one non-empty set $X \subset S$ such that $A_1 \cap X = \emptyset$ and $A_1 \cup X$ is the first component of a minimum cover of K. Let S_2 be a set of smallest cardinality among all candidates for X. Let (A_2, B_2) be a minimum cover with $A_2 = A_1 \cup S_2 = A_* \cup S_1 \cup S_2$. As before, B_2 is uniquely determined by A_2 and $B_2 \subset B_1$. Let $T_2 = B_1 \setminus B_2$. Since $|A_2| - |A_1| = |B_1| - |B_2|$, we have $|S_2| = |T_2|$. Since $B^* = T_1 \cup B_1$ and $T_2 = B_1 \setminus B_2$, we have $B^* = T_1 \cup T_2 \cup B_2$.

If we proceed in this manner, the process will stop for some k where

$$A_* \cup S_1 \cup S_2 \ldots \cup S_k = A^*. \tag{4}$$

At this point, (A^*, B_*) will be the resulting minimum cover. Furthermore, the T_i's satisfy

$$B^* = T_1 \cup T_2 \cup \ldots T_k \cup B_*. \tag{5}$$

In the intermediate stages of this process, (A_i, B_i) is a minimum cover for K for each $i \in \{1, 2, \ldots, k\}$ where

$$A_i = A_* \cup S_1 \cup S_2 \cup \ldots \cup S_i, \tag{6}$$

$$B_i = T_{i+1} \cup T_{i+2} \cup \ldots \cup T_k \cup B_*. \tag{7}$$

Equations (4) and (5) give the following decompositions of the vertex classes S and T.

$$S = A^* \bigcup (A^*)^c = A_* \cup S_1 \cup S_2 \ldots \cup S_k \bigcup (A^*)^c, \tag{8}$$

$$T = (B^*)^c \bigcup B^* = (B^*)^c \bigcup T_1 \cup T_2 \ldots \cup T_k \cup B_*. \tag{9}$$

The $k+2$ sets in the unions on the extreme right of both the above equations form a partition of S and T respectively. These partitions are unique except for a permutation of the S_i's having same cardinality, with the T_i's appropriately permuted.

With these definitions in place, we have the following result from [4].

▶ **Proposition 24.** *Let $K = (S, T, E)$ be a bipartite graph having a finite cover number. Then the Dulmage-Mendelsohn decomposition of K is given by the partition of $S \times T$ into the sets R_1, R_2, R_3 given by*

$$R_1 = (A_* \times (B^*)^c) \bigcup (S_1 \times T_1) \bigcup \dots \bigcup (S_k \times T_k) \bigcup ((A^*)^c \times B_*), \tag{10}$$

$$R_2 = (A_* \times B^*) \bigcup (A^* \times B_*) \bigcup_{i<j} (S_i \times T_j), \tag{11}$$

$$R_3 = ((A_*)^c \times (B^*)^c) \bigcup ((A^*)^c \times (B_*)^c) \bigcup_{i>j} (S_i \times T_j). \tag{12}$$

▶ **Example 25.** Consider the bipartite graph in Figure 3. It has vertex classes $S = \{P_1, P_2, P_3, P_4\}$ and $T = \{I_1, I_2, I_3\}$.

(i) As we noted in Example 17, $A_* = \emptyset, B^* = T$ and $A^* = \{P_1, P_2\}, B_* = \{I_3\}$.
(ii) As $(\{P_1\}, \{I_2, I_3\})$ is the only candidate for (A_1, B_1), we have $S_1 = \{P_1\}$ and $T_1 = \{I_1\}$.
(iii) As $(\{P_1, P_2\}, \{I_3\})$ is the only candidate for (A_2, B_2), we have $S_2 = \{P_2\}$ and $T_2 = \{I_2\}$.

The DM decomposition is given by

$$R_1 = \{(P_1, I_1), (P_2, I_2), (P_3, I_3), (P_4, I_3)\},$$
$$R_2 = \{(P_1, I_3), (P_2, I_3), (P_1, I_2)\},$$
$$R_3 = \{(P_3, I_1), (P_3, I_2), (P_4, I_1), (P_4, I_2), (P_2, I_1)\}$$

The graph has no edges in R_3. The edges in R_2 cannot appear in any maximum matching, as P_1 must be matched to I_1 and P_2 must be matched to I_2. The edges in R_1 appear in at least one maximum matching on the graph. ⌟

To visualize the DM decomposition, suppose that the vertices in S are ordered according to the partition in Equation (8), i.e. the vertices in A_* appear first, followed by vertices in $S_1, S_2, \dots, S_k$, and $(A^*)^c$. Similarly, suppose that the vertices in T are ordered according to the partition in Equation (9). Then the DM decomposition can be represented by Figure 4, where the rows correspond to vertices in T and the columns correspond to vertices in S. The admissible edges lie in blocks along the diagonal, the inadmissible edges lie above these blocks, and there are no edges below these blocks.

5.4 Computing the DM Decomposition

The DM decomposition of a bipartite graph K can be computed by finding a maximum matching M on K, then finding subsets of vertex classes unreachable from M via alternating paths, and finally by finding strongly connected components of the subgraph induced by the unreachable vertices (see [19] for details). Surprisingly, the DM decomposition is independent of the particular maximum matching chosen in the first step [19]. The component algorithms of the DM decomposition computation have worst-case running times which are polynomial in the number of graph vertices and edges.

We implemented the DM decomposition in Rust. Our code is available at `https://github.com/avras/cryptonote-analysis` under an MIT license. While an open source implementation of the DM decomposition already existed in CSparse [3], it was much

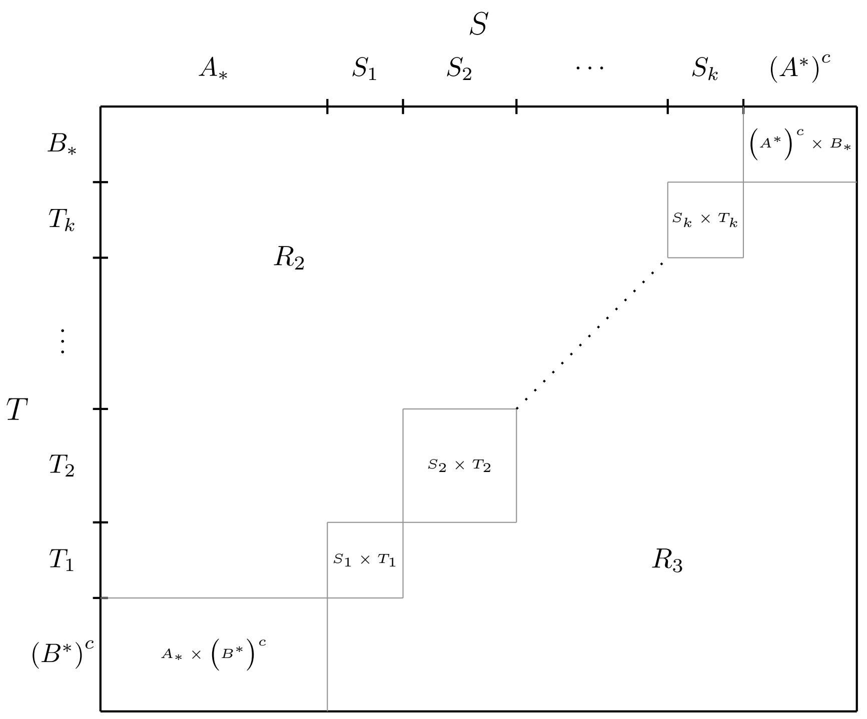


Figure 4 Visualization of the DM decomposition of a bipartite graph.

slower than the proprietary implementation in Matlab [11]. Our implementation of the DM decomposition has performance comparable to the Matlab implementation. Instructions on preparing the input data for our implementation are available at `https://www.respectedsir.com/cna`.

6 The DM Decomposition Finds All Closed Sets

By Theorem 6, the closed set attack is an optimal method for performing undeniable traceability analysis on CryptoNote transaction graphs. However, the naive method of finding closed sets by checking all subsets of the transaction rings [26, Appendix A] is not computationally feasible. The clustering algorithm for finding closed sets is guaranteed to find all closed sets of size 5 or less [26, Theorem 2]. While it does find closed sets with size more than 5, it is not guaranteed to find all closed sets.

In this section, we show that the DM decomposition finds all the closed sets in a CryptoNote transaction graph. As the DM decomposition can be computed in polynomial time, we obtain an efficient method to achieve the best possible undeniable traceability performance.

Let us establish/recall the following notation.

(i) Let $\mathcal{R}_h = \{R_1, R_2, \ldots, R_n\}$ be the multiset of all transaction rings which have appeared on the blockchain up to height h.

(ii) Let I_i be the key image corresponding to ring R_i.

(iii) Let $\mathcal{K}_h = \{I_1, I_2, \ldots, I_n\}$ be the set of all key images that have appeared on the blockchain up to height h.

(iv) Let $\mathcal{O}_h = \cup_{i=1}^{n} R_i = \{P_1, P_2, \ldots, P_m\}$ be the set of all keys (outputs) that have appeared in at least one transaction ring.

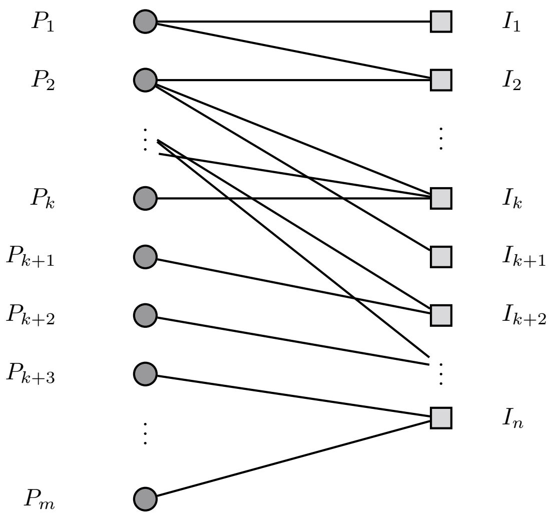


Figure 5 Transaction graph used in the proof of Theorem 26.

(v) Let $E \subseteq \mathcal{O}_h \times \mathcal{K}_h$ be the edge set of the transaction graph induced by the rings in $\mathcal{R}_h$. For $P \in \mathcal{O}_h$ and $I \in \mathcal{K}_h$, the edge (P, I) belongs to E if the output P appeared in the transaction ring used to create I.

(vi) Let C be a closed set of $\mathcal{R}_h$ having cardinality k. By Definition 4, there exist k transaction rings $R_{i_1}, R_{i_2}, \ldots, R_{i_k}$ in $\mathcal{R}_h$ such that $C = \cup_{j=1}^{k} R_{i_j}$. To describe this scenario briefly, we will say that the rings $R_{i_1}, R_{i_2}, \ldots, R_{i_k}$ *constitute* the closed set C. Furthermore, there exist k keys $P_{i_1}, \ldots, P_{i_k}$ in $\mathcal{O}_h$ such that $C = \{P_{i_1}, P_{i_2}, \ldots, P_{i_k}\}$.

(vii) Let $I_C = \{I_{i_1}, I_{i_2}, \ldots, I_{i_k}\}$ be the set of key images corresponding to the rings $R_{i_1}, R_{i_2}, \ldots, R_{i_k}$ that constitute C.

▶ **Theorem 26.** *Let $\mathcal{G}_h = (\mathcal{O}_h, \mathcal{K}_h, E)$ be the CryptoNote transaction graph induced by the rings in $\mathcal{R}_h$ and their corresponding key images. Let C be a closed set of $\mathcal{R}_h$ and let I_C be the set of key images corresponding to the rings that constitute C. Then $(C, \mathcal{K}_h \setminus I_C)$ is a minimum cover of $\mathcal{G}_h$. Conversely, if (A, B) is a minimum cover of $\mathcal{G}_h$ where $A \neq \emptyset$, then A is a closed set.*

Proof. Without loss of generality, we can assume that $R_1, R_2, \ldots, R_k$ are the rings that constitute C and $I_1, I_2, \ldots, I_k$ are the key images corresponding to these rings. Furthermore, let $C = \cup_{i=1}^{k} R_i = \{P_1, P_2, \ldots, P_k\}$.[3] By definition, $I_C = \{I_1, I_2, \ldots, I_k\}$.

Suppose we draw the bipartite graph induced by the blockchain history by listing $P_1, \ldots, P_k$ and $I_1, \ldots, I_k$ before the other vertices on each side. Let $P_{k+1}, \ldots, P_m$ be the other outputs in $\mathcal{O}_h$. Let $I_{k+1}, \ldots, I_n$ be the other key images in $\mathcal{K}_h$. Figure 5 illustrates this bipartite graph.

Since each key image in $I_1, I_2, \ldots, I_n$ corresponds to a unique true output on the left hand side, there exists a maximum matching of order n on this graph. By Proposition 21, minimum covers of this graph will also have size n. Note that every edge in the graph is incident on some element in $\{I_1, I_2, \ldots, I_n\}$. Thus $(\emptyset, \mathcal{K}_h) = (\emptyset, \{I_1, \ldots, I_n\})$ is a minimum cover of the graph.

[3] In general, the rings that constitute C and the corresponding keys and key images may not have indices $1, 2, \ldots, k$. But we can always relabel the elements in these sets to satisfy our assumption.

We claim that there are no edges between the key images $I_1, \ldots, I_k$ and the outputs $P_{k+1}, P_{k+2}, \ldots, P_m$. To see this, suppose there is an edge from I_j to P_l for some $j \in \{1, 2, \ldots, k\}$ and $l \in \{k+1, \ldots, m\}$. Then P_l must belong to the ring R_j as it is the only ring which contributes edges incident on I_j. This would mean P_l belongs to $\cup_{i=1}^{k} R_i = \{P_1, P_2, \ldots, P_k\}$, which is a contradiction as $l \geq k+1$. So all the edges incident on $I_1, \ldots, I_k$ must have an output from $P_1, \ldots, P_k$ on the other end.

The above argument shows that $(C, \mathcal{K}_h \setminus I_C) = (\{P_1, \ldots, P_k\}, \{I_{k+1}, \ldots, I_n\})$ is a minimum cover of the graph. Thus every closed set of $\mathcal{R}_h$ is the first member of a minimum cover of the transaction graph.

To prove the other direction, suppose that (A, B) is a minimum cover of the transaction graph where $A \neq \emptyset$. Let $B^c = \mathcal{K}_h \setminus B$ be the set of key images not in B. Suppose $B^c = \{I_{i_1}, I_{i_2}, \ldots, I_{i_l}\}$.

Since $(\emptyset, \mathcal{K}_h)$ is a minimum cover of the graph, every minimum cover must have size n. This implies that $|A| + |B| = n$. As $l = |B^c| = n - |B|$, the set A must have l outputs.

Since (A, B) is a cover of the transaction graph, every edge incident on key images in B^c must be covered by an output in A (as B can only cover edges incident on the key images in it). Each key image I_{i_j} in B^c is associated with a unique transaction ring R_{i_j} which contains the true output corresponding to it. The ring R_{i_j} is the set of outputs adjacent to I_{i_j} in the graph. We claim that R_{i_j} is a subset of A.

We prove our claim by contradiction. If there is a key P in R_{i_j} that is not contained in A, then the edge (P, I_{i_j}) exists but $P \notin A$ and $I_{i_j} \notin B$. This contradicts our assumption that (A, B) is a vertex cover.

Since the transaction ring R_{i_j} is a subset of A for every $I_{i_j} \in B^c$, we have $\cup_{j=1}^{l} R_{i_j} \subseteq A$. Furthermore, $\left|\cup_{j=1}^{l} R_{i_j}\right| \geq l$ because each of the l key images $I_{i_1}, I_{i_2}, \ldots, I_{i_l}$ has a unique true output in $\cup_{j=1}^{l} R_{i_j}$. Putting all this together, we have

$$l \leq \left| \bigcup_{j=1}^{l} R_{i_j} \right| \leq |A| = l. \tag{13}$$

We conclude that $A = \cup_{j=1}^{l} R_{i_j}$ and that $\left|\cup_{j=1}^{l} R_{i_j}\right| = l$, which proves that A is a closed set of $\mathcal{R}_h$. ◀

The above theorem says that finding all closed sets of a CryptoNote transaction graph is equivalent to finding all minimum covers of it. When the graph has only one minimum cover (i.e. when $A_* = A^*$), there is only one possible closed set. For the case when $A_* \neq A^*$, Dulmage and Mendelsohn proved the following theorem [4, Theorem 10] which shows that all the possible minimum covers of a bipartite graph can be calculated from the DM decomposition.

▶ **Theorem 27.** *Let $K = (S, T, E)$ be a bipartite graph having a finite cover number and more than one minimum cover. Let $A_*, B_*, S_1, S_2, \ldots, S_k, T_1, T_2, \ldots, T_k$ be as defined in Section 5.3.2. Let (A, B) be a minimum cover of K. Then exists a subset Λ of the index set $\{1, 2, \ldots, k\}$ such that*

$$A = \left(\bigcup_{i \in \Lambda} S_i \right) \cup A_*, \quad B = \left(\bigcup_{i \in \Lambda^c} T_i \right) \cup B_*.$$

This theorem implies that every minimum cover of the transaction graph can be recovered from the DM decomposition of the graph. This in turn implies that every closed set in the transaction graph can be found from the DM decomposition.

Table 1 Monero traceability of pre-RingCT rings by the clustering algorithm vs DM decomposition (up to block 1,541,236).

No. of mixins	No. of pre-RingCT rings	Traced by clustering algorithm	Traced by DM decomposition
0	12,209,675	12,209,675	12,209,675
1	707,786	625,641	625,641
2	2,941,525	1,779,134	1,779,446
3	1,345,574	952,855	952,862
4	972,457	451,959	451,981
5	143,793	74,186	74,186
6	366,894	202,360	202,360
7	12,361	4,296	4,296
8	9,148	3,506	3,506
9	6,396	2,178	2,178
$\geq$ 10	118,902	29,177	29,177
Total	18,834,511	16,334,967	16,335,308

7 DM Decomposition of the Monero Transaction Graph

We implemented the DM decomposition, the cascade attack, and the closed set attack in Rust. Our code is available at `https://github.com/avras/cryptonote-analysis` under an MIT license.

7.1 Empirical Analysis without Hard Fork Information

To evaluate the effectiveness of the DM decomposition in tracing transaction rings, we used the results obtained by the clustering algorithm of Yu et al. [26] on Monero as the benchmark. The latter results are the best results on Monero undeniable traceability which do not use information from hard forks.

Yu et al. considered Monero transactions contained in blocks with height up to 1,541,236 (March 30, 2018). This data set contains 23,164,745 transaction rings (each one contributing a key image) and 25,126,033 outputs. The corresponding bipartite graph has 58,791,856 edges. Out of the 23,164,745 transaction rings in the data set, 4,330,234 were RingCT rings and the remaining 18,834,511 were pre-RingCT rings.

Previous work [8], [17], [26], on Monero traceability has shown that RingCT transactions in Monero are immune to undeniable traceability attacks. The same observation holds for the DM decomposition approach. None of the 4,330,234 RingCT rings could be traced by the DM decomposition (when information from hard forks is not used). Table 1 compares the number of pre-RingCT transaction rings traced by the clustering algorithm and the DM decomposition. Each row in the table gives results for transaction rings which have a certain number of mixin outputs. The results for all transaction rings with 10 or more mixin outputs are combined in the row with label "$\geq$ 10".

All the 16,335,308 rings traced by the DM decomposition are associated with a set S_i with $|S_i| = 1$. The singleton set T_i corresponding to S_i has the key image of the output in S_i. As seen from the last row, the DM decomposition identifies 341 more traceable rings than the clustering algorithm. These new rings are only among the transaction rings having 2, 3, or 4 mixins.

Yu et al. report finding 3017 closed sets with sizes in the range 2 to 55. The DM decomposition is able to find 3045 closed sets with 3041 of them having sizes in the range 2 to 55. The remaining four closed sets have sizes 103, 106, 119, and 122. This discrepancy is due to the approximate nature of the clustering algorithm used by Yu et al. to find closed sets.

To check if the transactions which have appeared after block 1,541,236 have affected the traceability of RingCT rings, we computed the DM decomposition of the subgraph induced exclusively by RingCT transaction rings in all blocks up to height 2,530,000 (January 4, 2022).[4] This subgraph has 40,351,733 key images and 45,805,726 outputs with 409,626,277 edges between them. Let $\mathcal{K}$ be the set of all the key images in this subgraph. Its DM decomposition revealed only two minimum covers, $(\emptyset, \mathcal{K})$ and $(S_1, \mathcal{K} \setminus T_1)$ where $|S_1| = |T_1| = 5$. The set S_1 consists of RingCT outputs with indices 3890287, 3890288, 3890289, 3890290, and 3890291.

These five outputs were created by Wijaya et al. [22] in block 1,468,425. All of them were spent using the other four as mixins in five transaction rings in block 1,468,439 (Dec 17, 2017), to demonstrate that a set of outputs can be considered spent without relying on zero-mixin transactions. These five outputs are also marked as spent by the Monero blackball tool [13]. Thus, the DM decomposition of the Monero RingCT subgraph (using only main chain data) does not identify any new outputs as spent.

There were 37,038,237 RingCT transaction rings in the blocks with heights from 1,468,426 to 2,530,000. The five spent RingCT outputs were chosen as mixins in only 25 of these RingCT rings. Each of the 25 rings had at least 4 mixins and had their effective number of mixins reduced by one. Thus, the RingCT rings are mostly unaffected by the DM decomposition analysis.

The clustering algorithm was also able to identify the size 5 closed set. But it took 64 hours to finish running on our test machine while the DM decomposition could be computed in 4 hours.

7.2 Empirical Analysis using Hard Fork Information

To check the immunity of Monero RingCT transactions against the DM decomposition technique which incorporates hard fork information, we constructed a transaction graph using four different hard forks: Monero Original, MoneroV, Monero v7, and Monero v9. Table 2 gives the information regarding these forks where the last column contains the number of RingCT key images which appeared both in fork chain and the Monero main chain up to block height 2,530,000. While Monero Original and MoneroV were intentional hard forks created by developers who preferred a different design, the blocks on the Monero v7 and Monero v9 forks were unintentionally created by miners who were late in upgrading to the latest version of the main Monero client. This is the reason for the small number of blocks in the Monero v7 and Monero v9 forks.

We computed the performance of the DM decomposition technique on this graph up to Monero block height 2,530,000 (January 4, 2022). We found that 63,060 RingCT transaction rings out of 40,351,733 are undeniably traceable, i.e. only 0.15% of the RingCT rings are undeniably traceable. Note that the number of traceable RingCT rings is less than the total number of common RingCT key images shown in Table 2. This is because the appearance

[4] We could not try later block heights as the resulting transaction graphs were too large to fit in the memory of our test machine.

Table 2 Information about the four Monero hard forks.

Fork Name	Fork block	Number of blocks in fork	Number of common key images	Number of common RingCT key images
Monero Original	1,546,000	238,682	86,685	64,189
MoneroV	1,564,966	146,325	9,387	6,609
Monero v7	1,685,555	29	1,061	1,027
Monero v9	1,788,000	73	1,581	1,581

of key image in both the main chain and the fork chain does not imply traceability. If the transaction rings in both cases have more than one output in common, the true output being spent may not be identified.

The clustering algorithm was also able to trace the same 63,060 RingCT transaction rings. But it took 64 hours to finish while the DM decomposition took 4 hours. In fact, the cascade attack, which is the first step in the clustering algorithm (see Algorithm 4), was able to trace all these rings. The subsequent closed set search was fruitless as there were no closed sets in the transaction graph, except for the size 5 closed set induced by Wijaya et al. [22].

Readers interested in the effect of the DM decomposition analysis (using hard forks) on non-RingCT rings up to block height 2,530,000 can read the section at `https://www.respectedsir.com/cna/hardfork-nonringct.html` in our documentation.

8 Conclusion

We showed that the classical notion of the Dulmage-Mendelsohn decomposition of bipartite graphs gives an efficient implementation of the closed set attack, which is the optimal method for undeniable traceability in CryptoNote blockchains. Combining the DM decomposition with previously proposed methods for plausible traceability is an interesting direction for future work. We have released open source implementations of the DM decomposition, cascade attack, and clustering algorithm. We have also released the scripts used to generate all the empirical results in this paper along with detailed instructions on how to use them. We hope that these tools will be useful to other researchers, especially those working on methods for plausible traceability.

References

1 Sina Aeeneh, João Otávio Chervinski, Jiangshan Yu, and Nikola Zlatanov. New attacks on the untraceability of transactions in CryptoNote-style blockchains. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5, 2021. `doi:10.1109/ICBC51069.2021.9461130`.

2 Sherman S. M. Chow, Christoph Egger, Russell W. F. Lai, Ivy K. Y. Woo, and Viktoria Ronge. On sustainable ring-based anonymous systems. In *36th IEEE Computer Security Foundations Symposium*, 2023.

3 Timothy A. Davis. CSparse: A concise sparse matrix package. URL: `https://people.engr.tamu.edu/davis/suitesparse.html`.

4 A. L. Dulmage and N. S. Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958. `doi:10.4153/CJM-1958-052-0`.

5 Christoph Egger, Russell W. F. Lai, Viktoria Ronge, Ivy K. Y. Woo, and Hoover H.F. Yin. On defeating graph analysis of anonymous transactions. In Michelle Kerschbaum, Florian; Mazurek, editor, *Proceedings on Privacy Enhancing Technologies*, volume 2022 (3), pages 538–557, Warschau (Polen), 2022. Sciendo. `doi:10.56553/popets-2022-0085`.

6 Brandon Goodell. Perfect privacy or strong deniability? In *Monero Konferenco*, 2019. URL: `https://youtu.be/xicn4rdUj_Q`.

7 Abraham Hinteregger and Bernhard Haslhofer. An empirical analysis of Monero cross-chain traceability. In *Financial Cryptography and Data Security*, pages 150–157, 2019.

8 Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of Monero's blockchain. In *European Symposium on Research in Computer Security*, pages 153–173. Springer, 2017.

9 Joseph K Liu, Victor K Wei, and Duncan S Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *Australasian Conference on Information Security and Privacy*, pages 325–335. Springer, 2004.

10 L. Lovász and M.D. Plummer. *Matching Theory.* North-Holland, 1986.

11 dmperm: MATLAB function for Dulmage-Mendelsohn decomposition. URL: `https://in.mathworks.com/help/matlab/ref/dmperm.html`.

12 Monero Blackball Databases, 2021. Last Accessed: August 13, 2023. URL: `https://github.com/monero-blackball/monero-blackball-site`.

13 Monero Blackball Tool Code, 2023. Last Accessed: June 13, 2023. URL: `https://github.com/monero-project/monero/blob/master/src/blockchain_utilities/blockchain_blackball.cpp`.

14 Monero Original GitHub Repository, 2018. URL: `https://github.com/XmanXU/monero-original`.

15 Monero Scheduled Software Upgrades, 2020. Last Accessed: August 13, 2023. URL: `https://github.com/monero-project/monero/#scheduled-software-upgrades`.

16 MoneroV GitHub Repository, 2019. URL: `https://github.com/monerov/monerov`.

17 Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An empirical analysis of traceability in the Monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 2018(3):143–163, 2018. `doi:10.1515/popets-2018-0025`.

18 João Otávio Chervinski, Diego Kreutz, and Jiangshan Yu. Analysis of transaction flooding attacks against Monero. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–8, 2021. `doi:10.1109/ICBC51069.2021.9461084`.

19 Alex Pothen and Chin-Ju Fan. Computing the block triangular form of a sparse matrix. *ACM Trans. Math. Softw.*, 16(4):303–324, December 1990. `doi:10.1145/98267.98287`.

20 Viktoria Ronge, Christoph Egger, Russell W. F. Lai, Dominique Schröder, and Hoover H.F. Yin. Foundations of ring sampling. *Proceedings on Privacy Enhancing Technologies*, 2021:265–288, 2021. `doi:10.2478/popets-2021-0047`.

21 Nicolas van Saberhagen. CryptoNote v 2.0. White paper, 2013. URL: `https://cryptonote.org/whitepaper.pdf`.

22 Dimaz Ankaa Wijaya, Joseph Liu, Ron Steinfeld, and Dongxi Liu. Monero ring attack: Recreating zero mixin transaction effect. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1196–1201, 2018. `doi:10.1109/TrustCom/BigDataSE.2018.00165`.

23 Dimaz Ankaa Wijaya, Joseph Liu, Ron Steinfeld, Dongxi Liu, and Tsz Hon Yuen. Anonymity reduction attacks to Monero. In Fuchun Guo, Xinyi Huang, and Moti Yung, editors, *Information Security and Cryptology*, pages 86–100, Cham, 2019. Springer International Publishing.

24 Dimaz Ankaa Wijaya, Joseph K. Liu, Ron Steinfeld, Dongxi Liu, and Jiangshan Yu. On the unforkability of Monero. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, Asia CCS '19, pages 621–632, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3321705.3329823`.

25 J. Yu, M. H. A. Au, and P. Esteves-Verissimo. Re-thinking untraceability in the CryptoNote-style blockchain. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 94–106, 2019. `doi:10.1109/CSF.2019.00014`.

26 Zuoxia Yu, Man Ho Au, Jiangshan Yu, Rupeng Yang, Qiuliang Xu, and Wang Fat Lau. New empirical traceability analysis of CryptoNote-style blockchains. In *Financial Cryptography and Data Security*, pages 133–149, 2019.

Vector Commitments with Efficient Updates

Ertem Nusret Tas
Stanford University, CA, USA

Dan Boneh
Stanford University, CA, USA

Abstract

Dynamic vector commitments that enable local updates of opening proofs have applications ranging from verifiable databases with membership changes to stateless clients on blockchains. In these applications, each user maintains a relevant subset of the committed messages and the corresponding opening proofs with the goal of ensuring a succinct global state. When the messages are updated, users are given some global update information and update their opening proofs to match the new vector commitment. We investigate the relation between the size of the update information and the runtime complexity needed to update an individual opening proof. Existing vector commitment schemes require that either the information size or the runtime scale *linearly* in the number k of updated state elements. We construct a vector commitment scheme that asymptotically achieves both length and runtime that is *sublinear* in k, namely k^ν and $k^{1-\nu}$ for any $\nu \in (0,1)$. We prove an information-theoretic lower bound on the relation between the update information size and runtime complexity that shows the asymptotic optimality of our scheme. While in practice, the construction is not yet competitive with Verkle commitments, our approach may point the way towards more performant vector commitments.

2012 ACM Subject Classification Security and privacy

Keywords and phrases Vector commitments, stateless clients

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.29

Related Version *Full Version*: `https://arxiv.org/abs/2307.04085`

Funding This work was partially funded by NSF, DARPA, the Simons Foundation, and NTT Research. Additional support was provided by the Stanford Center for Blockchain Research. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

1 Introduction

A Vector Commitment (VC) scheme [14, 24, 13] enables a committer to succinctly commit to a vector of elements. Later, the committer can generate an *opening proof* to prove that a particular position in the committed vector is equal to a certain value. VCs have found many applications in databases and blockchains [26, 39] as they enable a storage system to only store a commitment to the vector instead of the entire vector. The data itself can be stored elsewhere along with opening proofs. In a multiuser system, every user might store only one position of the vector along with the opening proof for that position.

Dynamic VCs [13] are vector commitments that support updates to the vector. Suppose the committed vector is of length N and some $k < N$ positions in the vector are updated, so that a new vector commitment is published. Then, every user in the system will need to update their local opening proof to match the updated commitment, and this is done with the help of some global *update information* U that is broadcast to all users. This information is typically generated and published by a manager who maintains the entire vector. Applications of dynamic VCs include verifiable databases, zero-knowledge sets with

Table 1 Comparison of different VCs. $|U|$ denotes the length of the update information. T denotes the runtime of a single proof update. $|G|$ and $|H|$ denote the size of a single group element and a single hash value, respectively. T_G and T_f denote the time complexity of a single group operation and a single function evaluation for the hash function used by the VC. The last column PP is 'Y' if the proof update requires pre-processing to generate a global and fixed table of auxiliary data needed for proof updates.

Vector Commitment	$\|U\|$	T	PP
Merkle tree [25]	$\tilde{\Theta}(k)\ \|H\|$	$\tilde{O}(1)$	N
Hyperproofs [33]	$\tilde{\Theta}(k)\ \|G\|$	$\tilde{O}(1)$	N
Verkle tree [11]	$\tilde{\Theta}(k)\ \|G\|$	$\tilde{O}(1)\ \|H\|\ T_G$	Y
This work with $\nu \in [0,1]$	$\tilde{\Theta}(k^{\nu})\|H\|$	$\tilde{\Theta}(k^{1-\nu})\ T_f$	N
KZG commitments [19]	$\tilde{O}(1)$	$\tilde{\Theta}(k)\ T_G$	Y
RSA accumulators and VCs [7, 8]	$\tilde{O}(1)$	$\tilde{\Theta}(k)\ T_G$	N
Bilinear accumulators [27, 34]	$\tilde{O}(1)$	$\tilde{\Theta}(k)\ T_G$	N

frequent updates [13] and stateless clients for blockchains [9]. The challenge is to design a VC scheme that minimizes the size of the update information U as well as the computation work by each user to update their local opening proof.

For example, consider stateless clients on a blockchain as an important application for dynamic VCs. The state of the chain can be represented as a vector of length N, where position i corresponds to the state of account number i. Every user will locally maintain its own state (corresponding to some position in the vector) along with an opening proof that enables the user to convince a third party as to its current state. Whenever a new block is published, the state of the chain changes. In particular, suppose k out of the N positions in the vector need to be updated. The block proposer will publish the *update information* U along with the new block, and every user will update their opening proof to match the new committed state of the chain. Thus, users can ensure that their opening proofs are up to date with respect to the latest committed state of the chain.

We stress that in this application, the data being updated, namely the updated positions and diffs, is published as part of the block. The update information U only contains additional information that is needed to update the opening proofs. When we refer to the size of U, we refer to its size, excluding the updated data (i.e., excluding the updated positions and diffs).

In this paper, we investigate the trade-off between the length $|U|$ of the update information and the time complexity of proof updates. Dynamic VCs can be grouped into two categories in terms of these parameters (Table 1). Tree-based VCs [25, 33] enable users to update their proofs in time $O(\text{polylog}\, N)$. Each opening proof typically consists of $\text{polylog}\,(N)$ inner nodes, and the update information U contains the changes in the inner nodes affected by the message updates. Each user calculates its new opening proof by downloading the relevant inner nodes published as part of U. When k positions are updated, a total of $O(k \log(N))$ inner nodes in the tree are affected in the worst case. Thus, when each inner node has length $\Theta(\lambda)$, proportional to the security parameter λ, the update information consists of $O(k\log(N)\lambda)$ bits.

In contrast, algebraic VCs [19, 7, 8, 27, 34] enable users to update their opening proofs with only knowledge of the updated data. They do not require any additional update information U to be published beyond the indices and the "diffs" of the updated data. Thus, the length of the update information needed to update the opening proofs is $O(1)$. However, algebraic VCs typically require each user to read all of the changed messages and incorporate the effect of these changes on their proofs, resulting in $\Theta(k)$ work per proof update.

To summarize, while tree-based VCs support efficient calculation of the new opening proofs by publishing a large amount of update information, linear in k, algebraic VCs do not require any additional update information beyond the updated data, but suffer from a large runtime for proof updates, linear in k. We formalize the dichotomy of VCs in Section 3.

1.1 Our Results

We propose a family of VCs that can support *sublinear update*, where both the length $|U|$ of the update information and the complexity of proof updates are sublinear in k. More specifically, our VCs can attain $|U| = \Theta(k^\nu \lambda)$, $\nu \in (0,1)$, with a proof update complexity of $\Theta(k^{1-\nu})$ operations. Our candidate construction with sublinear update is a *homomorphic Merkle tree*, first developed by [29, 32], where each inner node can be expressed as a *sum* of the *partial digests* of the messages underneath (Section 4). The algebraic structure of these trees enable each user to calculate the *effect* of a message update on any inner node without reading other inner nodes or messages. We identify homomorphic Merkle tree constructions based on lattices, from the literature [30, 29, 32].

In Section 4, we provide the update algorithms (Alg. 1) for homomorphic Merkle trees, parameterized by $\nu \in (0,1)$. Our algorithm identifies a special subset of size $\tilde{\Theta}(k^\nu)$ of the inner nodes affected by the message updates, and publish their new values as U; so that the users need not calculate these values. These inner nodes are selected carefully to ensure that any inner node *outside* of U is affected by at most $\Theta(k^{1-\nu})$ updated messages. Thus, to modify its opening proof, each user has to calculate the partial digests of at most $\Theta(k^{1-\nu})$ updated messages per inner node within its proof (that consists of $\Theta(\log(N))$ inner nodes). Moreover, to calculate these partial digests, the user only needs the "diffs" of the updated messages. This brings the asymptotic complexity of proof updates to $\tilde{\Theta}(k^{1-\nu})$ operations, while achieving an update information size of $\tilde{\Theta}(k^\nu \lambda)$ as opposed to $\tilde{\Theta}(k\lambda)$ on Merkle trees using SHA256.

In Section 6, we prove an information theoretic lower bound on the size of the update information given an upper bound on the runtime complexity of proof updates. The bound implies the asymptotic optimality of our scheme with sublinear update. Its proof is based on the observation that if the runtime complexity is bounded by $O(k^{1-\nu})$, a user that wants to update its proof cannot read beyond $O(k^{1-\nu})$ updated messages. Then, to calculate the effect of the remaining $k - O(k^{1-\nu})$ messages on its opening proof, the user has to download parts of the structured update information U. Finally, to obtain the lower bound on $|U|$, we use Shannon entropy and lower bound the number of bits, namely $O(k^\nu \lambda)$, required to capture the total information that will be downloaded by the users; while maintaining the security of the VC with parameter λ.

1.2 Applications

We identify three main applications for VCs with sublinear update.

1.2.1 Stateless clients for Ethereum

Ethereum is the largest decentralized general purpose computation platform by market cap. Ethereum state (*e.g.*, user accounts) is currently stored in the form of a Merkle tree [5] and grows approximately by half every year [10]. Stateless clients [9, 10] were proposed to mitigate the problem of state bloat and prevent the state storage and maintenance from becoming a bottleneck for decentralization. Stateless clients maintain an opening proof to their account balances within the Ethereum state, thus can effortlessly prove the inclusion of their accounts within the latest state. This enables the other Ethereum clients to verify the transactions that come with opening proofs without having to download the full state and check the validity of the claimed account balances. Since block verification now requires downloading the proofs for the relevant state elements, Verkle trees [20, 11, 16] were proposed as a replacement for Merkle trees due to their short proof size.

Each new Ethereum block contains transactions that update the state elements and their opening proofs. Archival nodes and block producers still maintain the full state so that they can inform the stateless clients about their new opening proofs [10]. For this purpose, block producers must broadcast enough information to the clients over the peer-to-peer gossip network of Ethereum[1]. As minimizing the proof size was paramount to decentralizing verification for blocks, minimizing the update information size becomes necessary for decentralizing the role of the block producer who has to disseminate this information. However, reducing the length of the update information must not compromise the low overhead of stateless clients by requiring larger number of operations per proof update. Therefore, the ideal VC scheme for stateless clients must strike a delicate balance between the size of the update information and the runtime complexity of proof updates.

In Section 5, we provide the update algorithms for Verkle trees given their role in supporting stateless clients. We observe that Verkle trees do not support sublinear update, and fall under the same category as tree-based VCs with update information length $\tilde{\Theta}(k\lambda)$. Despite this fact, Verkle trees are highly practical in terms of updates. In Section 5.5, we estimate that the update information size after a typical Ethereum block does not exceed $|U| \approx 100$ kBytes (compared to the typical block size of < 125 kBytes). Moreover, each Verkle proof can be updated within approximately less than a second on commodity hardware. In contrast, even the most efficient homomorphic Merkle tree construction [32] requires an update information size of 110.88 MBytes and an update time of 32.6 seconds when the trade-off parameter ν is $1/2$, despite its asymptotic optimality (*cf.* Section 4.4). The large update information size is due to the lattice-based construction of these VCs. Despite their advantage in terms of concrete performance, unlike these lattice-based constructions, Verkle trees are not secure against quantum computers. Designing dynamic VCs that are asymptotically optimal, practically efficient and post-quantum resilient remains an open problem.

1.2.2 Databases with frequent membership changes

VCs with sublinear update can support databases with frequent membership changes. When a user first registers, a message is updated to record the membership of the user. The user receives this record and its opening proof, using which it can later anonymously prove its membership. When the user leaves the system, the message is once again updated to delete the record. In all these steps, membership changes result in updates to the opening proofs of other members. When these changes are frequent, it becomes infeasible to distribute new proofs after each change. VCs with sublinear update offer an alternative and efficient way to update the opening proofs of the users in the event of such changes.

[1] Block producers can enable the clients to succinctly verify the correctness of this information via SNARK proofs, thus still keeping the verification cost of blocks small.

1.3 Related Work

There are many VC constructions, each with different guarantees regarding the proof, commitment and public parameter sizes, verification time, updatability and support for subvector openings [14, 24, 13, 37, 29, 23, 21, 18, 17, 6, 40, 8, 34, 33, 19, 11] (cf [28] for an SoK of VCs). First formalized by [13], almost all VCs allow some degree of updatability. Whereas [29, 6, 8, 34] enable updating the commitment and the opening proofs with only the knowledge of the old and the new messages, most VCs require some structured update information beyond the messages when the users do not have access to the internal data structures. Among the lattice-based accumulators, vector commitments and functional commitments [18, 22, 31, 29, 32, 30, 38], constructions amenable to sublinear update are presented in [30, 29, 32, 31]. Homomorphic Merkle trees were formalized and instantiated by [30, 29, 32] in the context of streaming authenticated data structures and parallel online memory checking. The construction presented in [31, Section 3.4] offers an alternative VC with sublinear update as it is not a Merkle tree, yet has the property that each inner node can be expressed as a *sum* of the partial digests of individual messages.

An alternative design to support stateless clients is the agregatable subvector commitment (aSVC) scheme [36], which is a VC that enables aggregating multiple opening proofs into a succinct subvector proof. It enables each user to update its opening proof with the knowledge of the transactions in the blocks, and block producers to prove the validity of these transactions succinctly by aggregating the proofs submitted by the transacting users. As the scheme is based on KZG commitments, no update information is needed, yet, the update time complexity is linear in the number of transactions per block.

For dynamic accumulators that support additions, deletions and membership proofs, Camacho and Hevia proved that after k messages are deleted, $\Omega(k)$ bits of data must be published to update the proofs of the messages in the initial accumulated set [12, Theorem 1]. Their lower bound is information-theoretic and follows from a compression argument. Christ and Bonneau subsequently used a similar method to prove a lower bound on the global state size of a *revocable proof system* abstraction [15]. As revocable proof systems can be implemented by dynamic accumulators and vector commitments, their lower bound generalizes to these primitives, *i.e.*, after k messages are updated in a dynamic VC, at least $\Omega(k)$ bits of data must be published to update the opening proofs (*cf.* the full version of the paper [35, Appendix A] for the proof). They conclude that a stateless commitment scheme must either have a global state with linear size in the number of accounts, or require a near-linear rate of local proof updates. In our work, we already assume a linear rate of local proof updates, *i.e.*, after every Ethereum block or k messages in our parameterization, and that the message updates are publicized by the blockchain. We instead focus on the trade-off between the global structured update information size (beyond the published messages) and the runtime complexity of proof updates.

2 Preliminaries

2.1 Notation

We denote the security parameter by λ. An event is said to happen with *negligible probability*, if its probability, as a function of λ, is $o(1/\lambda^d)$ for all $d > 0$. An event happens with *overwhelming probability* if it happens except with negligible probability.

We denote the set $\{0, 1, 2, .., N-1\}$ by $[N]$. When $y = O(h(x) \operatorname{polylog}(x))$, we use the shorthand $y = \tilde{O}(h(x))$ (similarly for $\Theta(.)$ and $\tilde{\Theta}(.)$). The function $H(.)\colon \mathcal{M} \to \{0,1\}^\lambda$ represents a collision-resistant hash function. We denote the binary decomposition of an integer x by $\mathrm{bin}(x)$, and for $c > 2$, its base c decomposition by $\mathrm{bin}_c(x)$. A vector of N elements $(n_0, .., n_{N-1})$ is shown as $(n_i)_i$. The notation $\mathbf{x}[i{:}j]$ denotes the substring starting at the i^{th} index and ending at the j^{th} index within the sequence $\mathbf{x}$. The indicator function 1_P is equal to one if the predicate P is true, otherwise, it is zero. In the subsequent sections, k will be used to denote the number of updated messages.

For a prime p, let $\mathbb{F}_p$ denote a finite field of size p. We use $\mathbb{G}$ to denote a cyclic group of prime order p with generator g. The Lagrange basis polynomial for a given $x \in \mathbb{F}_p$ is denoted as $L_x(X)$:

$$L_x(X) = \prod_{\substack{i \in \mathbb{F}_p \\ i \neq x}} \frac{X - i}{x - i}$$

We will use $|G|$ and $|H|$ to denote the maximum size of the bit representation of a single group element and a single hash value respectively. We will use T_G and T_f to denote the time complexity of a single group operation and a single function evaluation for the hash functions in Section 4.1.

2.2 Vector Commitments

A vector commitment (VC) represents a sequence of messages such that each message can be proven to be the one at its index via an *opening proof.* A dynamic vector commitment allows updating the commitment and the opening proofs with the help of an *update information* when the committed messages are changed.

▶ **Definition 1** (from [13]). *Dynamic (updateable) vector commitments can be described by the following algorithms:*

$\textsc{KeyGen}(1^\lambda, N) \to pp$: *Given the security parameter λ and the size $N = \mathrm{poly}(\lambda)$ of the committed vector, the key generation algorithm outputs public parameters pp, which implicitly define the message space $\mathcal{M}$.*

$\textsc{Commit}_{pp}(m_0, .., m_{N-1}) \to (C, \mathsf{data})$: *Given a sequence of N messages in $\mathcal{M}$ and the public parameters pp, the commitment algorithm outputs a commitment string C and the data* data *required to produce the opening proofs for the messages. Here,* data *contains enough information about the current state of the VC's data structure (*i.e.*, the current list of committed messages) to help generate the opening proofs.*

$\textsc{Open}_{pp}(m, i, \mathsf{data}) \to \pi_i$: *The opening algorithm is run by the committer to produce a proof π_i that m is the i^{th} committed message.*

$\textsc{Verify}_{pp}(C, m, i, \pi_i) \to \{0, 1\}$: *The verification algorithm accepts (*i.e.*, outputs 1) or rejects a proof. The security definition will require that π_i is accepted only if C is a commitment to some $(m_0, .., m_{N-1})$ such that $m = m_i$.*

$\textsc{Update}_{pp}(C, (i, m_i)_{i\in[N]}, (i, m'_i)_{i\in[N]}, \mathsf{data}) \to (C', U, \mathsf{data}')$: *The algorithm is run by the committer to update the commitment C when the messages $(m_{i_j})_{j\in[k]}$ at indices $(i_j)_{j\in[k]}$ are changed to $(m'_{i_j})_{j\in[k]}$. The other messages in the vector are unchanged. It takes as input the old and the new messages, their indices and the data variable* data*. It outputs a new commitment C', update information U and the new data variable* data$'$*.*

$\textsc{ProofUpdate}_{pp}(C, p((i, m_i)_{i\in[N]}, (i, m'_i)_{i\in[N]}), \pi_j, m', i, U) \to \pi'_j$: *The proof update algorithm can be run by any user who holds a proof π_j for some message at index j and a (possibly) new message m' at that index. It allows the user to compute an updated proof π'_j (and the updated commitment C') such that π'_j is valid with respect to C', which contains*

m'_i, $i \in N$, as the new messages at the indices $i \in N$ (and m' as the new message at index i). Here, $p(.)$ specifies what portion of the old and the new messages is sufficient to update the opening proof. For instance, the proof update algorithm often does not need the old and the new messages in the open; but can carry out the proof update using only their differences. In this case, $p((i, m_i)_{i\in[N]}, (i, m'_i)_{i\in[N]}) = (i, m'_i - m_i)_{i\in N}$.

Correctness of a VC requires that $\forall N = \text{poly}(\lambda)$, for all honestly generated parameters $pp \leftarrow \text{KeyGen}(1^\lambda, N)$, given a commitment C to a vector of messages $(m_0, .., m_{N-1}) \in \mathcal{M}^N$, generated by Commit_{pp} (and possibly followed by a sequence of updates), and an opening proof π_i for a message at index i, generated by Open_{pp} or ProofUpdate_{pp}, it holds that $\text{Verify}_{pp}(C, m_i, i, \pi_i) = 1$ with overwhelming probability.

Security of a VC is expressed by the position-binding property:

▶ **Definition 2** (Definition 4 of [13]). *A VC satisfies position-binding if $\forall i \in [N]$ and for every PPT adversary $\mathcal{A}$, the following probability is negligible in λ:*

$$\Pr\left[\begin{smallmatrix} \text{Verify}_{pp}(C,m,i,\pi_i)=1\wedge \\ \text{Verify}_{pp}(C,m',i,\pi'_i)=1\wedge m\neq m' \end{smallmatrix} : \begin{smallmatrix} pp\leftarrow \text{KeyGen}(1^\lambda,N) \\ (C,m,m',\pi_i,\pi'_i)\leftarrow \mathcal{A}(pp) \end{smallmatrix}\right]$$

We relax the *succinctness* assumption of [13] and denote a value to be succinct in x if it is polylog(x).

Many VC constructions also satisfy the hiding property: informally, no PPT adversary $\mathcal{A}$ should be able to distinguish whether the VC was calculated for a vector $(m_0, .., m_{N-1})$ or a vector $(m'_0, .., m'_{N-1}) \neq (m_0, .., m_{N-1})$. In this work, we do not consider the hiding property since it is not explicitly required by our applications, and VCs can be made hiding by combining them with a hiding commitment [13].

2.3 KZG Polynomial Commitments

The KZG commitment scheme [19] commits to polynomials of degree bounded by ℓ using the following algorithms:

KeyGen$(1^\lambda, \ell) \rightarrow pp$**:** outputs $pp = (g, g^\tau, g^{(\tau^2)}, .., g^{(\tau^\ell)})$ as the public parameters, where g is the generator of the cyclic group $\mathbb{G}$ and τ is a trapdoor ($pp[i] = g^{\tau^i}$).

Commit$(pp, \phi(X)) \rightarrow (C, \mathsf{data})$**:** The commitment to a polynomial $\phi(X) = \sum_{i=0}^{\ell-1} a_i X^i$ is denoted by $[\phi(X)]$, and is computed as $[\phi(X)] = \prod_{i=0}^{\ell}(pp[i])^{a_i}$. The commitment algorithm outputs $C = [\phi(X)]$ and $\mathsf{data} = \phi(X)$.

Open$_{pp}(m, i, \mathsf{data}) \rightarrow \pi$**:** outputs the opening proof π_i that $\phi(i) = m$, calculated as the commitment to the quotient polynomial $(\phi(X) - \phi(i))/(X - i)$.

Verify(C, m, i, π) accepts if the pairing check $e(C/g^m, g) = e(\pi, pp[1]/g^i)$ holds.

We refer to [19] for the security analysis of this scheme.

2.4 Merkle Trees

Merkle Tree is a vector commitment using a collision-resistant hash function. In a Merkle tree, hashes of the committed messages constitute the leaves of a c-ary tree of height $h = \log_c(N)$, where each inner node is found by hashing its children. The depth of the root is set to be 0 and the depth of the leaves is $\lceil \log_c(N) \rceil$. The commitment function outputs the Merkle root as the commitment C and the Merkle tree as data. The opening proof for a message m_x at some index x is the sequence of $h(c-1)$ hashes consisting of the siblings of the inner nodes on the path from the root to the hash of the message m_x. We hereafter consider binary Merkle trees ($c = 2$) and assume $N = c^h = 2^h$ unless stated otherwise. Let $u_{b_0,b_1,..,b_{i-1}}$, $b_j \in \{0, 1\}$,

$j \in [i]$, denote an inner node at depth $i-1$ that is reached from the root by choosing the left child at depth j if $b_j = 0$ and the right child at depth j if $b_j = 1$ ($b_0 = \perp$ and $u_\perp$ is the root). By definition, for a message m_x at index x, $H(m_x) = u_{\perp,\mathrm{bin}(x)}$.

2.5 Verkle Trees

A Verkle tree [11, 16] is similar to a Merkle tree except that each inner node is calculated as the hash of the KZG polynomial commitment to its children. Let $b_j \in [c]$, $j = 1,..,h$, denote the indices of the inner nodes on the path from the root to a leaf at index x, $\mathrm{bin}_c(x) = (b_1,..,b_h)$, relative to their siblings. Define $f_{b_0,..,b_j}$, $j \in [h]$, as the polynomials determined by the children of the inner nodes on the path from the root to the leaf, where $f_{b_0} = f_\perp$ is the polynomial determined by the children of the root. Let $C_{b_0,..,b_j} = [f_{b_0,..,b_j}]$, $j \in [h]$, denote the KZG commitments to these polynomials. By definition, $u_{b_0,..,b_j} = H(C_{b_0,..,b_j})$, and the value of the polynomial $f_{b_0,..,b_j}$ at index b_{j+1} is $u_{b_0,..,b_{j+1}}$ for each $j \in [h]$. Here, $u_{b_0} = H(C_{b_0})$ is the root of the tree, and $u_{b_0,..,b_h}$ equals the hash $H(m_x)$ of the message at index x. For consistency, we define $C_{b_0,..,b_h}$ as m_x. For example, given $h = 3$ and $c = 4$, the inner nodes from the root to the message m_{14} have the indices $b_0 = 0$, $b_1 = 3$ and $b_2 = 2$, and they are committed by the polynomials $f_\perp$, $f_{\perp,0}$ and $f_{\perp,0,3}$ respectively.

The commitment function $\textsc{Commit}_{pp}(m_0,..,m_{N-1})$ outputs the root u_{b_0} as the commitment C and the Verkle tree itself as data.

The Verkle opening proof for the message m_x, $\mathrm{bin}(x) = (b_1,..,b_h)$, consists of two parts: (i) the KZG commitments $(C_{b_0,b_1},..,C_{b_0,..,b_{h-1}})$ on the path from the root to the message, and (ii) a Verkle multiproof. The goal of the Verkle multiproof is to show that the following evaluations hold for the inner nodes from the root to the message: $f_{b_0,..,b_j}(b_{j+1}) = u_{b_0,..,b_{j+1}} = H(C_{b_0,..,b_{j+1}})$, $j \in [h]$. It has two components: (i) the commitment $[g(X)]$ and (ii) the opening proof π' for the polynomial $h(X) - g(X)$ at the point $t = H(r, [g(X)])$, where

$$g(X) = \sum_{j=0}^{h-1} r^j \frac{f_{b_0,..,b_j}(X) - u_{b_0,..,b_{j+1}}}{X - b_{j+1}}, \qquad h(X) = \sum_{j=0}^{h-1} r^j \frac{f_{b_0,..,b_j}(X)}{t - b_{j+1}},$$

and $r = H(C_{b_0},..,C_{b_0,..,b_{h-1}}, u_{b_0,b_1},..,u_{b_0,..,b_h}, b_1,..,b_h)$. Thus, $\textsc{Open}_{pp}(m,i,\mathsf{data})$ outputs $((C_{b_0,b_1},..,C_{b_0,..,b_{h-1}}),([g(X)],\pi'))$.

To verify a Verkle proof $\pi = ((C_{b_0,b_1},..,C_{b_0,..,b_h}),(D,\pi'))$, algorithm $\textsc{Verify}_{pp}(C,m,x,\pi)$ first computes r and t using $u_{b_0,..,b_j} = H(C_{b_0,..,b_j})$, $j \in [h]$, and $u_{b_0,..,b_h} = H(m)$. Then, given the indices $\mathrm{bin}(x) = (b_1,..,b_h)$ and the commitments $(C_{b_0,b_1},..,C_{b_0,..,b_h})$, it calculates

$$y = \sum_{j=0}^{h-1} r^j \frac{C_{b_0,..,b_j}}{t - b_{j+1}} \qquad E = \sum_{j=0}^{h-1} \frac{r^j}{t - b_{j+1}} C_{b_0,..,b_j}.$$

Finally, it returns true if the pairing check $e(E - D - [g(X)], [1]) = e(\pi', [X - t])$ is satisfied.

As the degree c of a Verkle tree increases, size of the opening proofs and the runtime of the verification function decreases in proportion to the height $h = \log_c N$ of the tree. This enables Verkle trees to achieve a short opening proof size for large number of messages (as in the case of the Ethereum state trie) by adopting a large degree (*e.g.*, $c = 256$). In comparison, each Merkle proof consists of $(c-1)\log_c N$ inner nodes, which grows linearly as c increases.

3 Formalizing the Dichotomy of VCs

We first analyze the trade-off between the number of operations required by proof updates and the size of the update information U by inspecting different types of dynamic VCs.

Recall that the number of updated messages is $k \le N$.

3.1 Updating KZG Commitments and Opening Proofs

In the subsequent sections, we assume that each user has access to a dictionary of KZG commitments to the Lagrange basis polynomials $L_i(X)$, $i \in \mathbb{F}_p$, and for each polynomial, its opening proofs at each point $j \in \mathbb{F}_p$, $j < N$. With the help of this table, one can instantiate a KZG based VC to the messages $(m_i)_{i\in[N]}$, by treating them as the values of the degree N polynomial $\phi(X)$ at inputs $i \in \mathbb{F}_p$, $i < N$. We next analyze the complexity of the update information and the proof updates in this VC. The update and proof update algorithms are described in [35, Appendix F].

3.1.1 Update Information

Suppose the vector $(i, m_i)_{i\in[N]}$ is updated at some index i such that $m'_i \leftarrow m_i + \delta$ for some $\delta \in \mathbb{F}_p$. Then, the polynomial $\phi(X)$ representing the vector is replaced by $\phi'(X)$ such that $\phi'(X) = \phi(X)$ if $X \neq i$, and $\phi'(i) = \phi(i) + \delta$ at $X = i$. Thus, the new KZG commitment C' to $\phi'(X)$ is constructed from the commitment C to $\phi(X)$ as follows:

$$C' = [\phi'(X)] = [\phi(X) + \delta L_i(X)] = [\phi(X)][L_i(X)]^{\delta} = C \cdot [L_i(X)]^{\delta} = C \cdot [L_i(X)]^{m'_i - m_i}.$$

If the vector is modified at k different indices $i_1, ..., i_k$ from message m_{i_j} to m'_{i_j}, $j \in [k]$, then the new commitment $C' = [\phi'(X)]$ becomes

$$\begin{aligned}\left[\phi(X) + \sum_{j=1}^{k} (m'_{i_j} - m_{i_j}) L_{x_{i_j}}(X)\right] &= [\phi(X)] \prod_{j=1}^{k} [L_{i_j}(X)]^{(m'_{i_j} - m_{i_j})} \\ &= C \prod_{j=1}^{k} [L_{i_j}(X)]^{(m'_{i_j} - m_{i_j})}.\end{aligned}$$

Thus, the commitment can updated given only the old and the new messages at the updated indices, besides the table.

3.1.2 Proof Update

Let π_x denote the opening proof of a polynomial $\phi(X)$ at a point (x, m_x). When k messages are updated, the new opening proof π'_x can be found as a function of the old proof π_x and the opening proofs $\pi_{i_j,x}$ of the Lagrange basis polynomials $L_{i_j}(X)$, $j \in [k]$, at the index x ($m'_x = m_x + \sum_{j=1}^{k} (m'_{i_j} - m_{i_j}) \cdot 1_{x=i_j}$ is the new value of m_x after the k updates). Namely, π'_x is

$$\left[\frac{\phi'(X) - m_x - \sum_{j=1}^{k} \delta_j \cdot 1_{x=i_j}}{X - x}\right] = \pi_x \prod_{j=1}^{k} \left[\frac{L_{i_j}(X) - L_{i_j}(x)}{X - x}\right]^{m'_{i_j} - m_{i_j}} = \pi_x \prod_{j=1}^{k} \pi_{i_j,x}^{m'_{i_j} - m_{i_j}}$$

Thus, the proof can updated given only the old and the new messages at the updated indices, besides the table. The update information is set to be the empty set, *i.e.*, $U = \emptyset$.

3.1.3 Complexity

The size of the update information is constant, *i.e.*, $\tilde{\Theta}(1)$. Each user can update its proof after k accesses to the dictionary, and in the worst case, $\Theta(k \log |\mathcal{M}|) = \tilde{\Theta}(k)$ group operations as $\log(m'_i - m_i) \le \log |\mathcal{M}|$ for all $i \in [N]$.

3.2 Updating Merkle Trees and Opening Proofs

We next consider a Merkle tree and analyze the complexity of the update information size and the runtime for proof updates. A simple update scheme would be recalculating the new Merkle tree given all of the old messages or the old inner nodes of the Merkle tree, and the message updates. However, this implies a large complexity for the runtime of the proof update algorithm that scales as $\Omega(k)$ when users keep track of the inner nodes, and as $\Omega(N)$ when the users recalculate the tree from scratch at each batch of updates. Moreover, in many applications, the users do not have access to any messages or inner nodes besides those that are part of the Merkle proof held by the user. Hence, in the following sections, we describe update and proof update algorithms that reduce the runtime complexity of the proof updates at the expanse of larger update information (*cf.* the full version of the paper [35, Appendix F]).

3.2.1 Update Information

Suppose the vector $(i, m_i)_{i\in[N]}$ is updated at some index x, $(b_1, .., b_h) = \text{bin}(x)$, to m'_x. Then, the root $C = u_{b_0}$ and the inner nodes $(u_{b_0,b_1}, .., u_{b_0,b_1,..,b_h})$, $(b_1, .., b_h) = \text{bin}(i)$, must be updated to reflect the change at that index. Given the old inner nodes, the new values for the root and these inner nodes, denoted by $C' = u'_{b_0}$ and $(u'_{b_0,b_1}, .., u'_{b_0,b_1,..,b_h})$, are calculated recursively as follows:

$$u'_{b_0,b_1,..,b_h} \leftarrow H(m'_x),$$
$$u'_{b_0,b_1,..,b_j} \leftarrow \begin{cases} H(u'_{b_0,b_1,..,b_j,0}, u_{b_0,b_1,..,b_j,1})\text{if } b_{j+1} = 0,\ j < h \\ H(u_{b_0,b_1,..,b_j,0}, u'_{b_0,b_1,..,b_j,1})\text{if } b_{j+1} = 1,\ j < h \end{cases}$$

When the messages are modified at k different points i_j, $j \in [k]$, the calculation above is repeated k times for each update.

As the updated inner nodes are parts of the Merkle proofs, the update information consists of the new values at the inner nodes listed from the smallest to the largest depth in the canonical left to right order. For instance, $U = ((\bot, u'_\bot), (\bot 0, u'_0), (\bot 1, u'_1), (\bot 00, u'_{00}), (\bot 10, u'_{10}), ..)$ implies that the root $u_\bot$ and the inner nodes $u_{\bot 0}$, $u_{\bot 1}$, $u_{\bot 00}$ and $u_{\bot 10}$ were updated after k messages were modified at the leaves of the Merkle tree. We reference the updated inner nodes using their indices (*e.g.*, $U[b_0, b_1 .. b_j] = v$, when $(b_1 .. b_j, v) \in U$).

3.2.2 Proof Update

The Merkle proof π_x for a message at index x, $(b_1, .., b_h) = \text{bin}(x)$, is the sequence $(u_{\bar{b}_1}, u_{b_1,\bar{b}_2}, .., u_{b_1,b_2,..,\bar{b}_h})$. When k messages are updated, some of the inner nodes within the proof might have changed. A user holding the Merkle proof for index x can find the new values of these inner nodes by querying the update information with their indices.

3.2.3 Complexity

Upon receiving the update information U, each user can update its proof in $\Theta(\log^2(N) + |H|\log(N)) = \tilde{\Theta}(1)$ time by running a binary search algorithm to find the updated inner nodes within U that are part of its Merkle proof, and reading the new values at these nodes. Since modifying each new message results in $h = \log(N)$ updates at the inner nodes and some of the updates overlap, $|U| = \Theta(k\log(N/k)(\log(N) + |H|)) = \tilde{\Theta}(k)|H|$, as each updated inner node is represented by its index of size $\Theta(\log(N))$ and its new value of size $|H|$ in U.

3.3 Dichotomy of VCs

In the case of KZG commitments, $|U| = \tilde{\Theta}(1)$, and there is no information overhead on top of the message updates. For Merkle trees with an efficient proof update algorithm, $|U| = \tilde{\Theta}(k)|H|$, thus there is an extra term scaling in $\tilde{\Theta}(k)|H| = \tilde{\Theta}(k)\lambda$, since $|H| = \Omega(\lambda)$ for collision-resistant hash functions. In contrast, for KZG commitments, each user has to do $\tilde{\Theta}(k)$ group operations to update its opening proof; whereas in Merkle trees, each user can update its proof in $\tilde{\Theta}(1)$ time, which does not depend on k. Hence, KZG commitments outperform Merkle trees in terms of the update information size, whereas Merkle trees outperform KZG commitments in terms of the time complexity of proof updates. Table 1 generalizes this observation to a dichotomy between algebraic VC schemes and tree-based ones favoring shorter runtimes for proof updates. The algebraic and tree-based ones outperform each other in terms of the update information size and runtime complexity respectively.

4 Vector Commitments with Sublinear Update

We would like to resolve the separation in Table 1 and obtain a vector commitment, where both the size of the update information and the complexity of proof updates have a sublinear dependence on k. In particular, $|U| = \tilde{\Theta}(g_1(k)\lambda)$ in the worst case, and the proof update algorithm requires at most $\tilde{\Theta}(g_2(k))$ operations, where both $g_1(k)$ and $g_2(k)$ are $o(k)$. We say that such a VC supports *sublinear update*.

In this section, we describe a family of VCs with sublinear update, parameterized by the values $\nu \in (0,1)$ and characterized by the functions $(g_1, g_2) = (k^\nu, k^{1-\nu})$.

4.1 Homomorphic Merkle Trees

We first introduce homomorphic Merkle trees where messages placed in the leaves take values in a set $\mathcal{M}$. We will use two collision-resistant hash functions $\tilde{f}\colon \mathcal{D}\times\mathcal{D} \to \mathcal{R}$ and $f\colon \mathcal{M} \to \mathcal{R}$, where both $\mathcal{M}$ and $\mathcal{D}$ are vector spaces over some field $\mathbb{F}$, and $\mathcal{R}$ is an arbitrary finite set. We will also need an injective mapping $g : \mathcal{R} \to \mathcal{D}$, which need not be efficiently computable. We use $g^{-1} : \mathcal{D} \to \mathcal{R}$ to denote the inverse of g, meaning that $g^{-1}(g(x)) = x$ for all $x \in \mathcal{R}$. We require that g^{-1} be efficiently computable.

Now, for $j \in [h]$, where h is the height of the tree, every node $u_{b_0,..,b_j} \in \mathcal{D}$ of the homomorphic Merkle tree is characterized by the following expressions:

$$\begin{aligned}
&\text{a leaf node:} && g^{-1}(u_{b_0,\mathrm{bin}(i)}) = f(m_i) \\
&\text{an internal node:} && g^{-1}(u_{b_0,..,b_j}) = \tilde{f}(u_{b_0,..,b_j,0},\ u_{b_0,..,b_j,1}) \text{ for } j < h
\end{aligned}$$

The homomorphic property of the Merkle tree refers to the fact that there are efficiently computable functions

$$h_{i,j} : \mathcal{D} \to \mathcal{D} \qquad \text{for } i \in [N] \text{ and } j \in [h],$$

such that every inner node $u_{b_0,..,b_j} \in \mathcal{D}$ can be expressed as

$$\begin{aligned}
u_{b_0} &= \sum_{i\in[N]} h_{i,0}(m_i) \\
u_{b_0,..,b_j} &= \sum_{i\colon \mathrm{bin}(i)[0:j-1]=(b_1,..,b_j)} h_{i,j}(m_i).
\end{aligned}$$

We refer to the function $h_{i,j}$ as a *partial digest function* and refer to $h_{i,j}(m_i)$ as the *partial digest* of m_i. In a homomorphic Merkle tree, every internal node is the sum of the partial digests of the leaves under that node. We will show in Section 4.3 that each function $h_{i,j}$ can be expressed as an iterated composition of the functions f and $\tilde{f}$. Evaluating $h_{i,j}$ requires evaluating the functions f and $\tilde{f}$ exactly $h-j$ times.

Opening proof for a message consists of *both* children of the internal nodes on the path from the message to the root (as opposed to Merkle opening proofs that contain only the siblings of the internal nodes on the path). For instance, the opening proof for the message m_i at leaf index i, with $\mathrm{bin}(i) = (b_1, .., b_h)$, is $(i, (u_{b_0,..,b_j,0}, u_{b_0,..,b_j,1})_{j=0,..,h-1})$. Opening proofs are verified using the functions f and $\tilde{f}$ (not by using the functions $h_{i,j}$). To verify an opening proof $(i, (u_{b_0,..,b_j,0}, u_{b_0,..,b_j,1})_{j=0,..,h-1})$ for a message m_i with respect to the root u_{b_0}, the verifier checks if the following equalities hold:

$$\begin{aligned}
&\text{for the leaf:} && g^{-1}(u_{b_0,\mathrm{bin}(i)}) = f(m_i) \\
&\text{for the internal nodes:} && g^{-1}(u_{b_0,..,b_j}) = \tilde{f}(u_{b_0,..,b_j,0},\ u_{b_0,..,b_j,1}) \text{ for } j = h-1, .., 0.
\end{aligned}$$

If so, it accepts the proof, and otherwise it outputs reject.

As an example, consider a homomorphic Merkle tree that commits to four messsages m_0, m_1, m_2, m_3. Then, its root $u_\perp$ and inner nodes $u_{\perp,0}$, $u_{\perp,1}$, $u_{\perp,0,0}$, $u_{\perp,0,1}$, $u_{\perp,1,0}$, $u_{\perp,1,1}$ can be calculated as follows:

$$\begin{aligned}
u_\perp &= h_{0,0}(m_0) + h_{1,0}(m_1) + h_{2,0}(m_2) + h_{3,0}(m_3)\ ; & u_{\perp,0,0} &= h_{0,2}(m_0) \\
u_{\perp,0} &= h_{0,1}(m_0) + h_{1,1}(m_1)\ ; & u_{\perp,0,1} &= h_{1,2}(m_1) \\
u_{\perp,1} &= h_{2,1}(m_2) + h_{3,1}(m_3)\ ; & u_{\perp,1,0} &= h_{2,2}(m_2) \\
& & u_{\perp,1,1} &= h_{3,2}(m_3)
\end{aligned}$$

The opening proof for m_3 is given by $(3, ((u_{\perp,0}, u_{\perp,1}), (u_{\perp,1,0}, u_{\perp,1,1})))$, and verified by checking the following equations:

$$\begin{aligned}
&\text{for } u_{\perp,1,1}\text{:} && g^{-1}(u_{\perp,1,1}) = f(m_i) \\
&\text{for } u_{\perp,1}\text{:} && g^{-1}(u_{\perp,1}) = \tilde{f}(u_{\perp,1,0},\ u_{\perp,1,1}) \\
&\text{for } u_\perp\text{:} && g^{-1}(u_\perp) = \tilde{f}(u_{\perp,0},\ u_{\perp,1})
\end{aligned}$$

It now follows that when a message m_i is updated to m'_i, each inner node on the path from the leaf to the root can be updated from $u_{b_0,..,b_j}$ to $u'_{b_0,..,b_j}$ using the functions $h_{i,j}$ as follows:

$$u'_{b_0,..,b_j} \quad = \quad h_{i,j}(m'_i) + \sum_{\substack{x \neq i: \\ \mathrm{bin}(x)[0:j-1]=(b_1,..,b_j)}} h_{x,j}(m_x) \quad = \quad u_{b_0,..,b_j} + h_{i,j}(m'_i) - h_{i,j}(m_i)$$

When the partial digest functions are linear in their input, the expression $h_{i,j}(m'_i) - h_{i,j}(m_i)$ can be written as $h_{i,j}(m'_i) - h_{i,j}(m_i) = \mathrm{sign}(m'_i - m_i)h_{i,j}(|m'_i - m_i|)$. This lets us calculate the updated internal node using only the knowledge of the message diff $m'_i - m_i$. We provide examples of homomorphic Merkle tree constructions in Section 4.3 with linear partial digest functions $h_{i,j}$. Homomorphic Merkle proofs in these constructions consist of the two siblings of the inner nodes on the path from the proven message to the root and the vector commitment itself is given by $g^{-1}(b_\perp)$ (Section 4.3).

Unlike in Section 3.2, homomorphic Merkle trees enable calculating the new inner nodes after message updates using *only* the new and the old updated messages, in particular using only their difference. Hence, we can construct a tree that achieves the same complexity for the update information size as algebraic VCs, albeit at the expanse of the proof update complexity, *without requiring the users to keep track of the old messages or to calculate the tree from scratch given all messages.* This is in contrast to Merkle trees based on SHA256. The update and proof update algorithms of such a homomorphic Merkle tree with no structured update information and the same asymptotic complexity as algebraic VCs is described in the full version of the paper [35, Appendix B]. Since the homomorphic Merkle trees can achieve both extremes in terms of update information size and update runtime (Table 1), with a smart structuring of the update information, they can support sublinear update. We show how in the next subsection.

4.2 Structuring the Update Information

Algorithm 1 Algorithms for a homomorphic Merkle tree. Each user knows the total number of leaves N. The recursive algorithm UPDATENODE, parameterized by $\nu \in [0,1]$, takes an index as input, and checks if the new value of the node at that index is to be published as part of the update information U. If so, it appends the new value to U, and recursively calls itself on the children of the node. Not all of U and $(i, m'_i - m_i)_{i\in[N]}$ are passed to the proof update algorithm and its relevant parts are read selectively to keep the runtime at a minimum.

```
algorithm UPDATE($C, (i, m'_i - m_i)_{i\in[N]}, \mathcal{T}$)
    $U \leftarrow$ EMPTY()
    algorithm UPDATENODE(idx)
        $b_0, .., b_d \leftarrow$ idx
        $\mathcal{S} \leftarrow \{j \in [k] \colon 1_{\mathrm{bin}(i_j)[0:d]=(b_1,..,b_d)}\}$
        if $|\mathcal{S}| > k^{1-\nu}$
            $\mathcal{T}[b_0, .., b_d] \leftarrow \mathcal{T}[b_0, .., b_d] + \sum_{j\in\mathcal{S}} \mathrm{SIGN}(m'_{i_j} - m_{i_j}) h_{i_j,d}(|m'_{i_j} - m_{i_j}|)$
            $U[b_0, b_1, .., b_d] \leftarrow \mathcal{T}[b_0, b_1, .., b_d]$
            UPDATENODE($(b_0, .., b_d, 0)$)
            UPDATENODE($(b_0, .., b_d, 1)$)
        end if
    end algorithm
    UPDATENODE($(b_0)$)
    $C' \leftarrow \mathcal{T}[b_0]$
    return $(C', U, \mathcal{T})$
end algorithm
algorithm PROOFUPDATE($C, \pi_x, m'_x, x, U$)
    $\pi'_x \leftarrow \{\}$
    $(b_1, .., b_h) \leftarrow \mathrm{bin}(x)$
    for $d = h, .., 1$
        if $(b_0, b_1 .. \bar{b}_d) \in U$
            $\pi'_x[b_0, b_1 .. \bar{b}_d] \leftarrow U[b_0, b_1 .. \bar{b}_d]$
        else
            $\mathcal{S} \leftarrow \{j \in [k] \colon 1_{\mathrm{bin}(i_j)[0:d]=(b_1,..,\bar{b}_d)}\}$
            $\pi'_x[b_0, .., \bar{b}_d] \leftarrow \pi_x[b_0, .., \bar{b}_d] + \sum_{j\in\mathcal{S}} \mathrm{SIGN}(m'_{i_j} - m_{i_j}) h_{i_j,d}(|m'_{i_j} - m_{i_j}|)$
        end if
    end for
    return $\pi'_x$
end algorithm
```

We now describe the new update and proof update algorithms that enable homomorphic Merkle trees to achieve sublinear complexity as a function of the parameter ν (Alg. 1).

4.2.1 Update Information

When the messages $(i_j, m_{i_j})_{j\in[k]}$ change to $(i_j, m'_{i_j})_{j\in[k]}$, the update information U is generated recursively using the following algorithm:

1. Start at the root u_{b_0}. Terminate the recursion at an inner node if there are $k^{1-\nu}$ or less updated messages under that node.
2. If there are more than $k^{1-\nu}$ updated messages with indices $\geq N/2$, *i.e.*, under the right child, then publish the new right child of the root as part of U, and apply the same algorithm to the subtree rooted at the right child, with u_{b_0} and N replaced by $u_{b_0,1}$ and $N/2$ respectively.
3. If there are more than $k^{1-\nu}$ updated messages with indices less than $N/2$, *i.e.*, under the left child, then publish the new left child of the root as part of U, and apply the same algorithm to the subtree rooted at the left child, with u_{b_0} and N replaced by $u_{b_0,0}$ and $N/2$ respectively.

The new values of the inner nodes included in U are again listed from the smallest to the largest depth in the canonical left to right order.

4.2.2 Proof Update

When the messages $(i_j, m_{i_j})_{j\in[k]}$ are updated to $(i_j, m'_{i_j})_{j\in[k]}$, a user first retrieves the inner nodes within its Merkle proof that are published as part of the update information. It then calculates the non-published inner nodes within the proof using the partial digests. For instance, consider a user with the proof $(u_{\bar{b}_1}, u_{b_1,\bar{b}_2}, .., u_{b_1,b_2,..,\bar{b}_h})$ for some message m_x, $(b_1, .., b_h) = \text{bin}(x)$. To update the proof, the user first checks the update information U and replaces the inner nodes whose new values are provided by U: $u'_{b_1,..,\bar{b}_d} \leftarrow U[b_1 .. \bar{b}_d]$, $d \in [h]$, if $U[b_1 .. \bar{b}_d] \neq \perp$. Otherwise, the user finds the new values at the nodes $u_{b_1,..,\bar{b}_d}$, $d \in [h]$, using the functions $h_{x,d}$:

$$u'_{b_1,..,b_{d-1},\bar{b}_d} = u_{b_1,..,b_{d-1},\bar{b}_d} + \sum_{j\in[k]} 1_{\text{bin}(i_j)[:d]=(b_1,..,\bar{b}_d)} \left(\text{sign}(m'_{i_j} - m_{i_j}) h_{i_j,d}(|m'_{i_j} - m_{i_j}|)\right)$$

4.2.3 Complexity

Finally, we prove bounds on the complexity given by these algorithms:

▶ **Theorem 3.** *Complexity of the update information size and the runtime of proof updates are as follows: $g_1(k) = k^\nu$ and $g_2(k) = k^{1-\nu}$.*

Proof. Let $\mathcal{U}$ denote the subset of the inner nodes published by the algorithm as part of U such that no child of a node $u \in \mathcal{U}$ is published. Then, there must be over $k^{1-\nu}$ updated messages within the subtree rooted at each node $u \in \mathcal{U}$. Since there are k updated messages, and by definition of $\mathcal{U}$, the subtrees rooted at the nodes in $\mathcal{U}$ do not intersect at any node, there must be less than $k/k^{1-\nu} = k^\nu$ inner nodes in $\mathcal{U}$. Since the total number of published inner nodes is given by $\mathcal{U}$ and the nodes on the path from the root to each node $u \in \mathcal{U}$, this number is bounded by $k^\nu \log(N) = \tilde{\Theta}(k^\nu)$. Hence, $|U| = \Theta(k^\nu \log(N)(\log(N) + |H|)) = \tilde{\Theta}(k^\nu)|H| = \tilde{\Theta}(k^\nu)\lambda$, which implies $g_1(k) = k^\nu$.

For each inner node in its Merkle proof, the user can check if a new value for the node was provided as part of U, and replace the node if that is the case, in at most $\Theta(\log(N) + |H|)$ time by running a binary search algorithm over U. On the other hand, if the new value of a node in the proof is not given by U, the user can calculate the new value after at most $k^{1-\nu}\log(N)$ function evaluations. This is because there can be at most $k^{1-\nu}$ updated messages within the subtree rooted at an inner node, whose new value was not published as part of U. This makes the total time complexity of a proof update at most

$$\Theta(\log(N)(\log(N) + |H| + k^{1-\nu}\log(N)T_f)) = \tilde{\Theta}(k^{1-\nu})T_f,$$

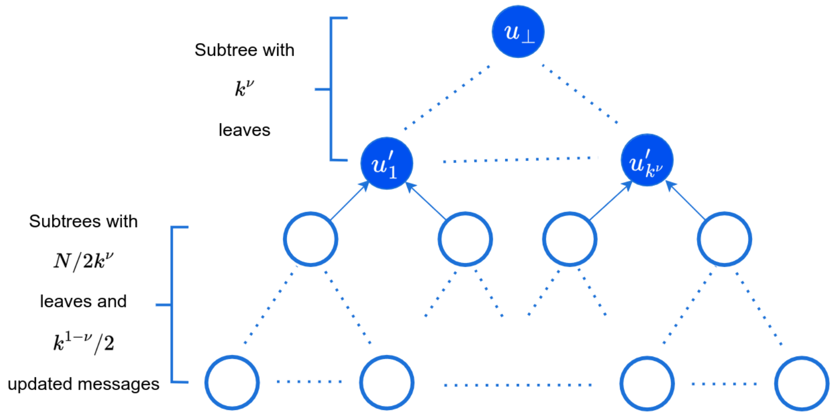


Figure 1 Homomorphic Merkle tree example. The new values of the inner nodes with solid blue color are published as part of the updated information.

which implies $g_2(k) = k^{1-\nu}$. ◀

To illustrate the proof above, consider the homomorphic Merkle tree in Figure 1 where k messages are updated. Suppose there are $k^{1-\nu}/2$ updated messages among the first $N/2k^\nu$ messages $m_0, .., m_{N/2k^\nu-1}$, another $k^{1-\nu}/2$ updated messages among the second $N/2k^\nu$ messages $m_{N/2k^\nu}, .. m_{2N/2k^\nu-1}$ and so on. In this case, the algorithm identifies the inner nodes within the subtree at the top of the tree (whose nodes are denoted in solid blue) and publishes their new values as part of the update information. This is because there are $k^{1-\nu}$ updated messages under each inner node and leaf of this subtree, denoted by u'_i, $i = 1, .., k^\nu$, whereas under the children of these leaf nodes there are less than $k^{1-\nu}$ updated messages. Thus, each user can update its opening proof by downloading the new values of the top $\log k^\nu$ inner nodes within its proof from the update information. There are at most $k^{1-\nu}/2$ updated messages under each of the remaining $\log N/k^\nu$ nodes in the proof; hence, the user can find their updated values in $\Theta(k^\nu \log N)$ time. Note that in this example, and in general when the updated messages are distributed uniformly among the leaves, the size of the update information becomes $\Theta(k^\nu)\lambda$ rather than $\Theta(k^\nu \log N)\lambda$.

4.3 Constructions for Homomorphic Merkle Trees

Homomorphic Merkle trees were proposed by [30, 29, 32]. They use lattice-based hash functions, and their collision-resistance is proven by reduction to the hardness of the gap version of the shortest vector problem (GAPSVP_γ), which itself follows from the hardness of the small integer solution problem. We next describe the construction introduced by [30], which is similar to those proposed by later works [29, 32]. Its correctness and security follow from [30, Theorem 4].

Let $L(\mathbf{M})$ denote the lattice defined by the basis vectors $\mathbf{M} \subset \mathbb{Z}_q^{k \times m}$ for appropriately selected parameters k, m, q, where $m = 2k \log q$. Consider vectors $u \in \{0, .., t\}^{k \log q}$, where t is a small integer. The (homomorphic) hash functions $f \colon \mathbb{Z}^{k \log q} \to L(\mathbf{M})$ and $\tilde{f} \colon \mathbb{Z}^{k \log q} \times \mathbb{Z}^{k \log q} \to L(\mathbf{M})$ used by [30] are defined as $f(x) = \mathbf{M}x$ and $\tilde{f}(x, y) = \mathbf{MU}x + \mathbf{MD}y$ respectively. Here, $\mathbf{U}$ and $\mathbf{D}$ are special matrices that double the dimension of the multiplied vector and shift it up or down respectively. The remaining entries are set to zero. For convenience, we define $\mathbf{L} = \mathbf{MU}$ and $\mathbf{R} = \mathbf{MD}$.

Since the domain and range of the hash functions are different, to ensure the Merkle tree's homomorphism, authors define a special mapping $g \colon \mathbb{Z}_q^k \to \mathbb{Z}_q^{k \log q}$ from the range of the hash functions to their domain. Here, $g(.)$ takes a vector $\mathbf{v} \in \mathbb{Z}_q$ as input and outputs *a* radix-2 representation for $\mathbf{v}$. However, as there can be many radix-2 representations of a vector, to help choose a representation that yields itself to homomorphism, authors prove the following result: for any $\mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_t \in \mathbb{Z}_q$, there exists *a short* radix-2 representation $g(.)$ such that $g(\mathbf{x}_1 + \mathbf{x}_2 + .. + \mathbf{x}_t \mod q) = b(\mathbf{x}_1) + b(\mathbf{x}_2) + .. + b(\mathbf{x}_t) \mod q \in \{0, .., t\}^{k \log q}$, where the function $b \colon \mathbb{Z}_q^k \to \{0,1\}^{k \log q}$ returns the binary representation of the input vector. This equality enables the mapping $g(.)$ to *preserve* the hash functions' original homomorphic property. Then, given an inner node $u_{b_0,..,b_j}$ as input, the homomorphic Merkle tree uses the short radix-2 representation $g(.)$ that enforces the following equality: $u_{b_0,..,b_j} = g(\mathbf{L}u_{b_0,..,b_j,0} + \mathbf{R}u_{b_0,..,b_j,1} \mod q) = b(\mathbf{L}u_{b_0,..,b_j,0}) + b(\mathbf{R}u_{b_0,..,b_j,1}) \mod q$. Finally, this enables calculating the value of each inner node as a sum of the partial digests $h_{i,j}(.)$ of the messages m_i under the node $u_{b_0,..,b_j}$ (*i.e.*, $(m_i)_{\text{bin}(i)[0:j]=(b_0,..,b_j)}$) as outlined in Section 4.1, i.e.,

$$u_{b_0,..,b_j} = \sum_{i \colon \text{bin}(i)[0:j-1]=(b_1,..,b_j)} h_{i,j}(m_i),$$

where $h_{i,j}(.)$ is expressed in terms of the bits $\text{bin}(i)[j{:}h-1] = (b'_1, .., b'_{h-j})$:

$$h_{i,j}(m_i) = f_{b'_1}(f_{b'_2}(.. f_{b'_{h-j}}(b(f(m_i)))))$$

Here, $f_0(.)$ and $f_1(.)$ are defined as $b(\mathbf{L}.)$ and $b(\mathbf{R}.)$ respectively. Since $b(.)$, binary expansion, is a linear operation and matrix multiplication is linear, $h_{i,j}(.)$ is linear in its input.

4.4 A Concrete Evaluation

Suppose the Ethereum state is persisted using the homomorphic Merkle tree construction of [29, 32] with the trade-off parameter $\nu = 1/2$. We next estimate the size of the update information and the proof update time after observing an Ethereum block with ERC20 token transfers. Suppose the block has the target size of 15 million gas [4], and each token transfer updates the balance of two distinct accounts stored at separate leaves of the homomorphic Merkle tree. Since each ERC20 token transfer consumes approximately $65,000$ gas, there are ~ 230 such transactions in the block, and the block updates $k = 460$ accounts.

Suppose the homomorphic Merkle tree has degree 2 and commits to $N = 256^3 = 2^{24}$ accounts. For comparison, $256^3 \approx 16.7$ million, matching in magnitude the total number of cumulative unique Ethereum addresses, which is 200 million as of 2023 [3]. Each opening proof consists of $2 \log(N) = 48$ inner nodes.

When 460 accounts are updated, in the worst case, the update information consists of $\lceil \sqrt{k} \rceil \log(N) = 528$ inner nodes. To evaluate its size, we use the parameters calculated by [32] for secure instantiations of the homomorphic Merkle trees from both their paper and [29]. Since the parameters for [29] result in a large inner node size on the order of hundreds of MBs, our evaluation takes the size of an inner node as that of [32], namely $|H| = 0.21$ MB (which is equal to the key size in [32]). This implies an update information size of $|U| = 110.88$ MBytes and an opening proof size of $|\pi| = 10.08$ MBytes.

As for update time, in the worst case, each user has to calculate the partial digests of 44 updated messages at each height of the homomorphic Merkle tree, *i.e.*, the effect of these updated messages on each inner node of its opening proof. Calculating the partial digest of a message at height h measured from the leaves requires h evaluations of the hash function. This implies a proof update complexity of $2 \sum_{i=0}^{\log N - 1} i \min(\lceil \sqrt{k} \rceil, 2^i) = 11,900$ hash evaluations. To find numerical upper bounds for the update time, we use the hash

Table 2 For different trade-off points between the update information size and proof update complexity, parameterized by ν, the table shows the number of published inner nodes $\lceil k^{\nu} \rceil \log(N)$, the total update information size $\lceil k^{\nu} \rceil \log(N)|H|$, the number of hash function evaluations per proof update $2\sum_{i=0}^{\log N-1} i \min(\lceil k^{1-\nu} \rceil, 2^i)$ and the proof update time $2\sum_{i=0}^{\log N-1} i \min(\lceil k^{1-\nu} \rceil, 2^i) T_f$. There are $N = 2^{24}$ accounts in total, $k = 460$ updates at the accounts, the inner nodes have size $|H| = 0.21$ Mbytes, and a hash function evaluation takes $T_f = 2.74$ ms.

ν	# inner nodes	$\|U\|$ (MBytes)	# hash evaluations	Time (s)
0	1	0.21	$227,972$	624.6
1/4	120	25.20	$52,284$	143.3
1/2	528	110.88	$11,900$	32.6
3/4	2400	504.00	2750	7.54
1	11040	2318.40	552	1.51

function evaluation times, namely $T_f = 26.84$ and $T_f = 2.74$ ms, published by [32] for both the hash function in [29] and their new and more performant function (these times are for commodity hardware; *cf.* [32] for the details). This gives an upper bound of 319.4 and 32.6 seconds for the update time using the hash functions in [29] and [32] respectively.

Based on the benchmarks for the practical hash function introduced in [32], Table 2 compares the number of published inner nodes $\lceil k^{\nu} \rceil \log(N)$, the total update information size $\lceil k^{\nu} \rceil \log(N)|H|$ (assuming that the size of each inner node is $|H|$ upper bounded by 0.21 MBytes), the number of hash function evaluations per proof update $2\sum_{i=0}^{\log N-1} i \min(\lceil k^{1-\nu} \rceil, 2^i)$ and the proof update time $2\sum_{i=0}^{\log N-1} i \min(\lceil k^{1-\nu} \rceil, 2^i) T_f$ (assuming that each hash evaluation takes less than $T_f = 2.74$ ms) at $\nu = 0, 1/4, 1/2, 3/4, 1$. The degree of the homomorphic Merkle tree and the opening proof size are fixed at 2 and 48 inner nodes ($|\pi| = 10.08$) respectively.

5 Updating Verkle Trees and Opening Proofs

We now describe the update and proof update functions for Verkle trees (see Algs. 2 and 3 in the full version of the paper [35, Section 5] for update and proof update algorithms respectively). Since Verkle trees were proposed to support stateless clients, we describe an update scheme that minimizes the runtime complexity of proof updates and does not require the users to download the updated messages or have access to old inner nodes. As Verkle trees do not support sublinear update, we numerically estimate the size of the update information and the complexity of proof updates in Section 5.5.

5.1 Update Information

Suppose the vector $(i, m_i)_{i \in [N]}$ is modified at some index x, $(b_1, .., b_h) = \text{bin}(x)$ to be m'_x. Since each inner node is the hash of a KZG commitment, the new inner nodes $u'_{b_0,..,b_j} = H(C'_{b_0,..,b_j})$, $j \in [h]$, can be found as a function of the old commitments at the nodes and the powers of the Lagrange basis polynomials as described in Section 3.1:

$$C'_{b_0,..,b_h} \leftarrow m'_x, \qquad C'_{b_0,..,b_j} \leftarrow C_{b_0,..,b_j} [L_{b_{j+1}}]^{(u'_{b_0,...,b_{j+1}} - u_{b_0,...,b_{j+1}})}$$

When k messages are updated, the above calculation is repeated k times for each update.

Update information U consists of the new values of the KZG commitments on the path from the updated messages to the Verkle root akin to the Merkle trees, ordered in the canonical top-to-bottom and left-to-right order.

5.2 Verkle Proofs

Let π_x denote the Verkle proof of some message m_x at index x, $(b_1, .., b_h) = \text{bin}(x)$: $\pi_x = ((C_{b_0,b_1}, .., C_{b_0,..,b_{h-1}}), ([g(X)], \pi))$. We define π_x^f as the opening proof for index x within polynomial f. We observe that the commitment $[g(X)]$ and the proof π can be expressed as functions of the opening proofs of the inner nodes $u_{b_0,b_1}, .., u_{b_0,..,b_h}$ at the indices $b_1, .., b_h$ within the polynomials $f_{b_0}, .., f_{b_0,..,b_{h-1}}$, respectively. Namely, $[g(X)]$ is

$$\left[\sum_{j=0}^{h-1} r^j \frac{f_{b_0,..,b_j}(X) - u_{b_0,..,b_{j+1}}}{X - b_{j+1}}\right] = \prod_{j=0}^{h-1} \left[\frac{f_{b_0,..,b_j}(X) - u_{b_0,..,b_{j+1}}}{X - b_{j+1}}\right]^{r^j} = \prod_{j=0}^{h-1} \left(\pi_{b_{j+1}}^{f_{b_0,..,b_j}}\right)^{r^j}$$

Similarly, the opening proof $\pi = \pi_t^{(h-g)}$ for index t within the polynomial $h(X) - g(X)$ can be expressed as follows (for details, see the full version of the paper [35, Appendix E]):

$$\left[\frac{h(X) - g(X) - (h(t) - g(t))}{X - t}\right] = \prod_{j=0}^{h-1} \left[\frac{f_{b_0,..,b_j}(X) - u_{b_0,..,b_{j+1}}}{X - b_{j+1}}\right]^{\frac{r^j}{t - b_{j+1}}}$$
$$= \prod_{j=0}^{h-1} \left(\pi_{b_{j+1}}^{f_{b_0,..,b_j}}\right)^{\frac{r^j}{t - b_{j+1}}}$$

We assume that each user holding the Verkle proof π_x for some index x, $(b_1, .., b_h) = \text{bin}(x)$, also holds the opening proofs $\pi_{b_{j+1}}^{f_{b_0,..,b_j}}$, $j \in [h]$, *in memory.* As we will see in the next section, the user also holds the KZG commitments at the children of the inner nodes on the path from the root to the message m_x, *i.e.* $C_{b_0,..,b_j,i}$ for all $j \in [h]$ and $i \in [c]$ *in memory.* These opening proofs and KZG commitments are not broadcast as part of any proof; however, they are needed for the user to locally update its Verkle proof after message updates.

5.3 Proof Update

When the messages $(i_j, m_{i_j})_{j \in [k]}$ are updated to $(i_j, m'_{i_j})_{j \in [k]}$, to calculate the new Verkle proof π'_x, the user must obtain the new commitments $C'_{b_0}, .., C'_{b_0,..,b_{h-1}}$ on the path from the root to message m_x, the new commitment $[g'(X)]$ and the new opening proof π' for the polynomial $h'(X) - g'(X)$ at index $t' = H(r', [g'(X)])$. Message updates change the commitments at the inner nodes, which in turn results in new polynomials $f_{b_0,..,b_j}$, $j \in [h]$. Suppose each polynomial $f_{b_0,..,b_j}$, $j \in [h]$, is updated so that

$$f'_{b_0,..,b_j}(X) = f_{b_0,..,b_j}(X) + \sum_{i=0}^{c-1} (f'_{b_0,..,b_j}(i) - f_{b_0,..,b_j}(i)) L_i(X),$$

where, by definition, $f'_{b_0,..,b_j}(i) - f_{b_0,..,b_j}(i) = u'_{b_0,..,b_j,i} - u_{b_0,..,b_j,i} = H(C'_{b_0,..,b_j,i}) - H(C_{b_0,..,b_j,i})$. Then, given the new and the old commitments $(C_{b_0,..,b_j,i}, C'_{b_0,..,b_j,i})$ for $i \in [c]$ and $j \in [h]$, the table of Lagrange basis polynomials, and using the technique in Section 3.1, the new opening proofs $\tilde{\pi}_{b_{j+1}}^{f_{b_0,..,b_j}}$ after the message updates can be computed as follows for $j \in [h]$:

$$\tilde{\pi}_{b_{j+1}}^{f_{b_0,..,b_j}} = \pi_{b_{j+1}}^{f_{b_0,..,b_j}} \prod_{i=0}^{c-1} \left[\frac{L_i(X) - L_i(b_{j+1})}{X - b_{j+1}}\right]^{(H(C'_{b_0,..,b_j,i}) - H(C_{b_0,..,b_j,i}))},$$

where $\left[\frac{L_i(X)-L_i(b_{j+1})}{X-b_{j+1}}\right]$ is the opening proof of the Lagrange basis polynomial $L_i(X)$ at index b_{j+1}. Once the new opening proofs are found, the new commitment $[g'(X)]$ and the new proof π' become

$$[g'(X)] = \prod_{j=0}^{h-1} \left(\tilde{\pi}_{b_{j+1}}^{f_{b_0,\dots,b_j}}\right)^{r'^j}, \qquad \pi' = \prod_{j=0}^{h-1} \left(\tilde{\pi}_{b_{j+1}}^{f_{b_0,\dots,b_j}}\right)^{\frac{r'^j}{t'-b_{j+1}}}$$

where $r' = H(C'_{b_0,b_1}, .., C'_{b_0,..,b_{h-1}}, u'_{b_0,b_1}, .., u'_{b_0,..,b_h}, b_1, .., b_h)$ and $t' = H(r', [g'(X)])$. Note that both r' and t' can be calculated by the user given the new KZG commitments $C'_{b_0,..,b_j,i}$ for all $i \in [c]$ and $j \in [h]$.

Finally, to retrieve the new KZG commitments $C'_{b_0,..,b_j,i}$ for all $i \in [c]$ and $j \in [h]$, the user inspects the commitments published as part of the update information U: $C'_{b_0,b_1,..,b_{j-1},i} \leftarrow U[b_0, b_1, .., b_{j-1}, i]$ if $U[b_0, b_1, .., b_{j-1}, i] \neq \perp$ and $C'_{b_0,b_1,..,b_{j-1},i} \leftarrow C_{b_0,b_1,..,b_{j-1},i}$ otherwise, for all $i \in [c]$ and $j \in [h]$.

In Verkle trees, the user cannot calculate the effect of an updated message on an arbitrary inner node without the knowledge of the inner nodes on the path from the message to the target node. For instance, suppose $U[b_0, b_1, .., b_{j-1}, i] = \perp$ for some $i \in [c]$ and $j \in [h]$, and the user wants to calculate the effect of an update from m_x to m'_x on $C'_{b_0,..,b_{j-1},i,\tilde{b}_{j+1},..,\tilde{b}_h}$, $\text{bin}(x) = (b_1, .., b_{j-1}, i, \tilde{b}_{j+1}, .., \tilde{b}_h)$ and $\tilde{b}_j = i$. Then, for each $\ell \in \{j, .., h-1\}$, the user have to find

$$C'_{b_0,..,\tilde{b}_j,..,\tilde{b}_h} \leftarrow m'_x$$
$$C'_{b_0,..,\tilde{b}_j,..,\tilde{b}_\ell} \leftarrow C_{b_0,..,\tilde{b}_j,..,\tilde{b}_\ell}[L_{\tilde{b}_{\ell+1}}]^{(u'_{b_0,..,\tilde{b}_j,..,\tilde{b}_{\ell+1}} - u_{b_0,..,\tilde{b}_j,..,\tilde{b}_{\ell+1}})},$$

where $C'_{b_0,..,\tilde{b}_j,..,\tilde{b}_\ell}$ are the commitments on the path from the target commitment $C_{b_0,b_1,..,b_{j-1},i}$ to the message m_x. Hence, the user has to know the original commitments on the path from the message to the target commitment, *i.e.*, keep track of inner nodes, which contradicts with the idea of stateless clients. This shows the necessity of publishing all of the updated inner nodes as part of the update information.

5.4 Complexity

Suppose each KZG commitment is of size $|G|$ and each hash $H(C)$ of a KZG commitment, *i.e.* each inner node, has size $|H|$. Then, updating a single message results in one update at each level of the Verkle tree and requires $\Theta(h|H|)$ group operations. Thus, when k messages are updated, the new Verkle root can be found after $\Theta(kh|H|)$ group operations. As U consists of the published KZG commitments at the inner nodes and their indices, $|U| = \Theta(k \log_c(N)(\log(N) + |G|)) = \tilde{\Theta}(k)|G|$, which implies $g_1(k) = k$.

The user can replace each KZG commitment at the children of the inner nodes from the root to its message in $\Theta(\log(N) + |G|)$ time by running a binary search algorithm over U. Since there are ch such commitments to be updated, *i.e.*, $C_{b_0,..,b_j,i}$, $i \in [c]$ and $j \in [h]$, updating these commitments takes $\Theta(ch(\log(N) + |G|)) = \tilde{\Theta}(1)$ time.

Upon obtaining the new commitments $C'_{b_0,..,b_{j-1},i}$, $i \in [c]$, $j \in [h]$, with access to the table of Lagrange basis polynomials, the user can update each opening proof $\pi_{b_{j+1}}$ (for the function $f_{b_0,..,b_j}$), $j \in [h]$, with $\Theta(c|H|)$ group operations. Since there are h such proofs, updating them all requires $\Theta(ch|H|)$ group operations. Given the new proofs, computing the new commitment $[g'(X)]$ and proof π' requires $\Theta(h|H|)$ group operations. This makes the total complexity of updating a Verkle proof $\Theta(ch + 2h)|H|T_G + \Theta(ch(\log_c(N) + |G|))$. For a constant c and $h = \log_c(N)$, this implies a worst-case time complexity of $\tilde{\Theta}(1)|H|T_G$ for Verkle proof updates, *i.e.*, $g_2(k) = 1$.

5.5 A Concrete Evaluation

We now estimate the size of the update information and the number of group operations to update an opening proof after observing an Ethereum block consisting of ERC20 token transfers. As in Section 4.4, suppose the block has the target size of 15 million gas [4], and each token transfer updates the balance of two distinct accounts stored at separate leaves of the Verkle tree. Then, there are ~ 230 such transactions in the block, and the block updates $k = 460$ accounts. We assume that the Verkle tree has degree 256 (*cf.* [11]) and commits to 256^3 accounts as in Section 4.4. Then, each proof consists of 2 KZG commitments, $C_{\perp,b_1}$ and $C_{\perp,b_1,b_2}$ and a multiproof consisting of the commitment $[g(X)]$ and proof π'. These components are elements of the pairing-friendly elliptic curve BLS12_381 and consist of $|G| = 48$ bytes [11]. This implies a proof size of $(\log_c(N) + 1)|G| = 192$ bytes (excluding the message at the leaf and its hash value; adding those makes it 272 bytes).

When 460 accounts are updated, in the worst-case, the update information has to contain $k \log_c(N)(\log(N) + |G|) = 460 \times 3 \times (24 + 48)$ Bytes, *i.e.*, 99.4 kBytes. This is comparable to the size of the Ethereum blocks, which are typically below 125 kBytes [2]. Hence, even though the update information of Verkle trees is linear in k, it does not introduce a large overhead beyond the block data. Note that the runtime of the proof updates are constant and do not scale in the number of updated messages k, or the Ethereum block size.

On the other hand, in the worst case, an opening proof can be updated after $c \log(c)|H| + 2\log_c(N)|H|$ group operations. Then, with $|H| = 256$, the number of bits output by SHA256, as many as $c \log_c(N)|H| + 2\log_c(N)|H| = (c+2)\log_c(N)|H| = 774 \times 2256 \approx 200{,}000$ elliptic curve multiplications might have to be made. Following the benchmarks published in [1] for the specified curve, these operations can take up to $(c+2)\log_c(N)$ 0.000665471 ns $= 0.52$ seconds on commodity hardware, given a runtime of 665471 nanoseconds per exponentiation of a group element with a message hash value. This is again comparable to the 12 second inter-arrival time of Ethereum blocks.

Table 3 compares the Verkle proof size $|\pi| = (\log_c(N) + 1)|G|$, update information size $|U| = k \log_c(N)(\log_c N + |G|)$, the upper bound $(c+2)\log_c N|H|$ on the number of group operations needed for a single proof update and the estimated time it takes to do these operations on a commodity hardware for different values of c, the Verkle tree degree, while keeping the number of accounts and the updated accounts fixed at 2^{24} and 460 respectively. The table shows the trade-off between the Verkle proof and update information size on one size and update complexity on the other.

Comparing Table 3 with Table 2 shows that the Verkle tree with any given degree c, $1 < c \leq 256$, significantly outperforms the existing homomorphic Merkle trees in Section 4.4 in terms of almost all of proof size, update information size and proof update time.

6 Lower Bound

Finally, we prove the optimality of our VC scheme with sublinear update by proving a lower bound on the size of the update information given an upper bound on the complexity of proof updates. The lower bound is shown for VCs that satisfy the following *proof-binding* property. It formalizes the observation that for many dynamic VCs (*e.g.*, Merkle trees [25], Verkle trees [11], KZG commitments [19], RSA based VCs [8]) including homomorphic Merkle trees (*cf.* the full version of the paper [35, Section 6]), the opening proof for a message at some index can often act as a commitment to the vector of the remaining messages.

Table 3 For different values of the tree degree c, the table shows the Verkle proof size which is $|\pi| = (\log_c(N) + 1)|G|$; the update information size which is $|U| = k \log_c(N)(\log(N) + |G|)$; the number of group operations for a single proof update which is $(c+2)\log_c(N)|H|$; and the estimated time for a single proof update. We use $N = 2^{24}$ accounts in total, $k = 460$ updates at the accounts, a group element size of $|G| = 48$ bytes, and a hash size of $|H| = 32$ bytes.

c	$\lvert\pi\rvert$ (Bytes)	$\lvert U\rvert$ (kBytes)	# Group Operations	Time (s)
2	1200	794.9	24,576	0.064
4	628	397.4	18,432	0.048
16	336	198.7	27,648	0.072
64	240	132.5	67,584	0.18
256	192	99.4	198,144	0.52

▶ **Definition 4.** *A VC scheme is said to be* proof-binding *if the following probability is negligible in λ for all PPT adversaries $\mathcal{A}$:*

$$\Pr\left[\begin{array}{r} \textsc{Verify}_{pp}(C,m_{i^*},i^*,\pi)=1 \\ \wedge\textsc{Verify}_{pp}(C',m_{i^*},i^*,\pi)=1 \end{array} : \begin{array}{r} pp\leftarrow\textsc{KeyGen}(1^\lambda,N); \\ \pi,m_{i^*},(m_0,..,m_{i^*-1},m_{i^*+1},..,m_{N-1}), \\ (m'_0,..,m'_{i^*-1},m'_{i^*+1},..,m'_{N-1})\leftarrow\mathcal{A}(pp); \\ (m_0,..,m_{i^*-1},m_{i^*+1},..,m_{N-1}) \\ \neq(m'_0,..,m'_{i^*-1},m'_{i^*+1},..,m'_{N-1}); \\ \textsc{Commit}_{pp}(m_0,..,m_{i^*-1},m_{i^*},m_{i^*+1},..,m_{N-1})=C; \\ \textsc{Commit}_{pp}(m'_0,..,m'_{i^*-1},m_{i^*},m'_{i^*+1},..,m'_{N-1})=C' \end{array}\right]$$

In the definition above, the opening proof while committing to the vector of remaining messages, need not be of the same format as the original VC. For instance, for RSA accummulators, the opening proof is itself an RSA accummulator, whereas for Merkle trees, the opening proof is not a Merkle tree root, but contains a sequence of inner nodes and the index of the opened message. Nevertheless, the proof and the message together act as a commitment to the vector of remaining messages.

The proof-binding property is distinct from the position-binding (security) property of VCs: whereaas position-binding states the difficulty of opening the VC to two different messages at the same index, proof-binding implies the difficulty of creating two VCs with different messages that open to the *same* message at some index i with the exact *same* proof.

We next state the main lower bound for proof-binding VCs.

▶ **Theorem 5.** *Consider a dynamic and proof-binding VC such that for every PPT adversary $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{array}{c} \textsc{Verify}_{pp}(C,m,i,\pi_i)=1\wedge \\ \textsc{Verify}_{pp}(C,m',i,\pi'_i)=1 \ \wedge\ m\neq m' \end{array} : \begin{array}{c} pp\leftarrow\textsc{KeyGen}(1^\lambda,N) \\ (C,m,m',\pi_i,\pi'_i)\leftarrow\mathcal{A}(pp) \end{array}\right] \leq e^{-\Omega(\lambda)}.$$

Then, for this VC, if $g_2(k) = O(k^{1-\nu})$, then $g_1 = \Omega(k^\nu)$ for all $\nu \in (0, 1)$.

Proof of Theorem 5 is given in the full version of the paper [35, Section 6].

▶ Remark 6. Theorem 5 shows that the update information length scales as $\tilde{\Theta}(k^\nu\lambda)$ when the runtime complexity for proof updates is $\tilde{\Theta}(k^{1-\nu})$ and the error probability for the security of the VC is $e^{-\Omega(\lambda)}$ for a PPT adversary. When the error probability is just stated to be negligible in λ, then the same proof can be used to show that the update information length must scale as $\Omega(k^\nu \operatorname{polylog}(\lambda))$ for any polynomial function of $\log(\lambda)$.

References

1 benchmarks-bls-libs. URL: `https://github.com/AlexiaChen/benchmarks-bls-libs`.

2 Ethereum average block size chart. URL: `https://etherscan.io/chart/blocksize`.

3 Ethereum cumulative unique addresses. URL: `https://ycharts.com/indicators/ethereum_cumulative_unique_addresses`.

4 Gas and fees. URL: `https://ethereum.org/en/developers/docs/gas/`.

5 State trie. URL: `https://ethereum.github.io/execution-specs/autoapi/ethereum/frontier/trie/index.html`.

6 Shashank Agrawal and Srinivasan Raghuraman. Kvac: Key-value commitments for blockchains and beyond. In *ASIACRYPT (3)*, volume 12493 of *Lecture Notes in Computer Science*, pages 839–869. Springer, 2020.

7 Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 1993.

8 Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 561–586. Springer, 2019.

9 Vitalik Buterin. The stateless client concept, 2017. URL: `https://ethresear.ch/t/the-stateless-client-concept/172`.

10 Vitalik Buterin. A state expiry and statelessness roadmap, 2021. URL: `https://notes.ethereum.org/@vbuterin/verkle_and_state_expiry_proposal`.

11 Vitalik Buterin. Verkle trees, 2021. URL: `https://vitalik.ca/general/2021/06/18/verkle.html`.

12 Philippe Camacho and Alejandro Hevia. On the impossibility of batch update for cryptographic accumulators. In *LATINCRYPT*, volume 6212 of *Lecture Notes in Computer Science*, pages 178–188. Springer, 2010.

13 Dario Catalano and Dario Fiore. Vector commitments and their applications. In *Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2013.

14 Dario Catalano, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 433–450. Springer, 2008.

15 Miranda Christ and Joseph Bonneau. Limits on revocable proof systems, with applications to stateless blockchains. *IACR Cryptol. ePrint Arch.*, page 1478, 2022. Appeared in the International Conference on Financial Cryptography and Data Security 2023.

16 Dankrad Feist. Pcs multiproofs using random evaluation, 2021. URL: `https://dankradfeist.de/ethereum/2021/06/18/pcs-multiproofs.html`.

17 Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. Pointproofs: Aggregating proofs for multiple vector commitments. In *CCS*, pages 2007–2023. ACM, 2020.

18 Mahabir Prasad Jhanwar and Reihaneh Safavi-Naini. Compact accumulator using lattices. In *SPACE*, volume 9354 of *Lecture Notes in Computer Science*, pages 347–358. Springer, 2015.

19 Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

20 John Kuszmaul. Verkle trees, 2018. URL: `https://math.mit.edu/research/highschool/primes/materials/2018/Kuszmaul.pdf`.

21 Russell W. F. Lai and Giulio Malavolta. Subvector commitments with application to succinct arguments. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 530–560. Springer, 2019.

22 Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 1–31. Springer, 2016.

23 Benoît Libert, Somindu C. Ramanna, and Moti Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In *ICALP*,

volume 55 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

24 Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 499–517. Springer, 2010.

25 Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987.

26 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: `https://bitcoin.org/bitcoin.pdf`.

27 Lan Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2005.

28 Anca Nitulescu. Sok: Vector commitments. URL: `https://www.di.ens.fr/~nitulesc/files/vc-sok.pdf`.

29 Charalampos Papamanthou, Elaine Shi, Roberto Tamassia, and Ke Yi. Streaming authenticated data structures. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2013.

30 Charalampos Papamanthou and Roberto Tamassia. Cryptography for efficiency: Authenticated data structures based on lattices and parallel online memory checking. *IACR Cryptol. ePrint Arch.*, page 102, 2011.

31 Chris Peikert, Zachary Pepin, and Chad Sharp. Vector and functional commitments from lattices. In *TCC (3)*, volume 13044 of *Lecture Notes in Computer Science*, pages 480–511. Springer, 2021.

32 Yi Qian, Yupeng Zhang, Xi Chen, and Charalampos Papamanthou. Streaming authenticated data structures: Abstraction and implementation. In *CCSW*, pages 129–139. ACM, 2014.

33 Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. Hyperproofs: Aggregating and maintaining proofs in vector commitments. In *USENIX Security Symposium*, pages 3001–3018. USENIX Association, 2022.

34 Shravan Srinivasan, Ioanna Karantaidou, Foteini Baldimtsi, and Charalampos Papamanthou. Batching, aggregation, and zero-knowledge proofs in bilinear accumulators. In *CCS*, pages 2719–2733. ACM, 2022.

35 Ertem Nusret Tas and Dan Boneh. Vector commitments with efficient updates. *arXiv:2307.04085*, 2023. URL: `https://arxiv.org/abs/2307.04085`.

36 Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. In *SCN*, volume 12238 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2020.

37 Alin Tomescu, Yu Xia, and Zachary Newman. Authenticated dictionaries with cross-incremental proof (dis)aggregation. *IACR Cryptol. ePrint Arch.*, page 1239, 2020.

38 Hoeteck Wee and David J. Wu. Succinct vector, polynomial, and functional commitments from lattices. *IACR Cryptol. ePrint Arch.*, page 1515, 2022.

39 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2014. URL: `https://files.gitter.im/ethereum/yellowpaper/VIyt/Paper.pdf`.

40 Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew K. Miller. hbACSS: How to Robustly Share Many Secrets. In *NDSS*. The Internet Society, 2022.

Time Is Money: Strategic Timing Games in Proof-Of-Stake Protocols

Caspar Schwarz-Schilling
Ethereum Foundation, Berlin, Germany

Fahad Saleh
Wake Forest University, Winston Salem, NC, USA

Thomas Thiery
Ethereum Foundation, Lyon, France

Jennifer Pan
Jump Crypto, Chicago, IL, USA

Nihar Shah
Jump Crypto, Chicago, IL, USA

Barnabé Monnot
Ethereum Foundation, Berlin, Germany

Abstract

We propose a model suggesting that rational consensus participants may play *timing games*, and strategically delay their block proposal to optimize MEV capture, while still ensuring the proposal's inclusion in the canonical chain. In this context, ensuring economic fairness among consensus participants is critical to preserving decentralization. We contend that a model grounded in rational consensus participation provides a more accurate portrayal of behavior in economically incentivized systems such as blockchain protocols. We empirically investigate timing games on the Ethereum network and demonstrate that while timing games are worth playing, they are not currently being exploited by consensus participants. By quantifying the marginal value of time, we uncover strong evidence pointing towards their future potential, despite the limited exploitation of MEV capture observed at present.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases blockchain, proof-of-stake, game theory, maximal extractable value

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.30

Acknowledgements The authors acknowledge helpful discussions and comments from Francesco d'Amato and Anders Elowsson. We also appreciate the significant contributions of Mike Neuder in obtaining the necessary data for this study.

1 Introduction

Consensus protocols are typically evaluated based on their ability to maintain liveness and safety [11], referring to the regular addition of new transactions to the output ledger in a timely manner, and to the security of confirmed transactions remaining in their positions within the ledger. However, beyond liveness and safety, blockchain protocols require fairness of economic outcomes amongst consensus participants to preserve decentralization. More specifically, a protocol should be designed to maximize profitability of honest participation, defined as adherence to the prescribed rules. Otherwise, a deviating participant may outcompete their honest peers, leading to centralization of the validation set over time and security implications for consensus itself.

However, the advent of Maximal Extractable Value (MEV) frustrates such fairness goals. MEV is defined as the value that consensus participants, in their duties as block proposers, accrue by selectively including, excluding and ordering user transactions [14, 6]. This concept has significant implications for the security of consensus protocols. For systems predominantly based on transaction fee rewards, increased variance in miner rewards may result in consensus instability [12]. Similarly, it was argued that a rational actor issuing a *whale transaction* with an abnormally large transaction fee can convince peers to fork the current chain, further destabilizing consensus [22]. As such, understanding and mitigating the impact of MEV on the security and fairness of blockchain networks has become a central concern of protocol designers [30].

Potential MEV accrues over time as users submit transactions and the value of the set of pending transactions increases for the block proposer. As a consequence, time is valuable to consensus participants, a feature obviated by the assumption of honest behavior in previous models of consensus. However, we argue that protocols who wish to preserve properties such as economic fairness amongst consensus participants must assume some share of rational consensus participation. In particular, the effects of MEV on the consensus participants' incentives must be better understood.

In this paper, we investigate the possibility for block proposers to delay their block proposal as long as possible while ensuring they become part of the canonical chain, aiming to maximize MEV extraction. The reader may note that in Proof-of-Work (PoW)-based leader selection protocols, delaying a proposal bears the risk of losing to a competing block proposer. PoW protocols exhibit an inherent racing condition that prevents these types of strategic delay deviations, or at least make them unprofitable in expectation. Thus, we investigate the implications of MEV on the incentives of consensus participants, particularly block proposers, in a Proof-of-Stake (PoS) context. More specifically, we consider propose-vote type of PoS protocols, where in each consensus round, one leader proposes a block, and a committee of consensus participants is selected in-protocol to vote on the acceptance of that block. This effectively grants block proposers a short-lived monopoly as the only valid proposer for some given round. During this time interval they can attempt to strategically deviate from their assigned block proposal time and delay the release of their block as long as possible in order to extract more MEV, while still ensuring that a sufficient share of attesters see the block in time to vote it into the canonical chain. This behavior leads to an environment in which honest validators earn less than their deviating counterparts, resulting in stake centralization and second-order effects for consensus stability.

Related Work

To the best of our knowledge, *timing games* have not been formally analyzed in previous literature on Proof-of-Stake. Selfish mining [19], studied in the context of Proof-of-Work, relies on appropriately timing the release of a block, in order to waste computation of honest miners and earn an outsize share of the rewards. Our timing games are also concerned with strategic behavior to capture a larger share of the total available rewards to consensus participants, yet do not feature the same dynamics as selfish mining in Proof-of-Work, since participants in many PoS-based consensus mechanisms are given a fixed time interval in which to perform their duties.

The security of PoS-based mechanisms has been discussed in terms of chain growth [18] or focusing on the safety and liveness properties of hybrid protocols such as Gasper [10, 24]. The economics literature has also examined Proof-of-Stake security with respect to particular attacks such as the double-spending attack [26] and 51% attacks [20]. Separately, incentive considerations in the presence of MEV led to the discovery of severe attacks on the Gasper consensus [28, 25] and protocol changes to address such attacks [15, 16, 17].

Our Contributions

Our work models the value of time to consensus participants and explores the potential emergence of timing games in Proof-of-Stake protocols. By understanding the strategic behavior of consensus participants within this model, we gain insights into how these dynamics affect the robustness of consensus protocols to exogenous incentives, and ultimately fairness.

- Despite initial pessimism regarding the existence of equilibria in timing games [23], we formally show how to sustain equilibrium behavior, where it is individually irrational for proposers to deviate from a schedule enforced by attesters, and reward-sharing is fair among participants (Sections 2 and 3).
- We then investigate whether such timing games might occur in real-world systems (namely, the Ethereum network), using a large, granular data set recording the MEV offered to block proposers over time. We show incidental deviations from the honest protocol specification, highlighting the feasibility of timing games, yet we do not conclude on the existence of intentional deviations from honest behavior (Section 4).

2 Model

We model an infinite horizon game among *block proposers* and *attesters*. Time is partitioned into slots $n \in \mathbb{N}$, each of time length $\Delta > 0$. Each slot n has a block proposer n and a unit measure of attesters $A_n = \{A_{(i,n)}\}_{i\in[0,1]}$ where $A_{(i,n)}$ refers to the ith attester within slot n.[1]

The game evolves as follows:

- At the beginning of slot n, proposer n acts by deciding whether to build on top of the block of proposer $n-1$ and also when to release their own block. More formally, proposer n selects $\phi_n \in \{0,1\}$ and $t_n \geq n \cdot \Delta$ where $\phi_n = 1$ ($\phi_n = 0$) refers to proposer n (not) building on top of the block of proposer $n-1$ and t_n denotes the time at which proposer n releases their own block. Note that we specify that proposer n cannot release their block before the start of slot n (i.e., $t_n \geq n \cdot \Delta$) but that they release their block after the end of the slot.
- After proposer n acts, all slot n attesters act simultaneously. In particular, attester $A_{(i,n)}$ decides whether to attest to the block of proposer n and also the time to release their attestation. More formally, attester $A_{(i,n)}$ select $\nu_{(i,n)} \in \{0,1\}$ and $\tau_{(i,n)} \geq n \cdot \Delta$ where $\nu_{(i,n)} = 1$ ($\nu_{(i,n)} = 0$) refers to attester $A_{(i,n)}$ (not) attesting to proposer n's block and $\tau_{(i,n)}$ refers to the time that they release their attestation. Notably, attester $A_{(i,n)}$ can attest to the block of proposer n only if they receive the block before releasing their attestation. We let $\delta_{n,(i,n)} \sim exp(\theta^{-1})$ refer to the random time required for the block of proposer n to reach attester $A_{(i,n)}$ where $\theta > 0$ denotes the average communication time across the network and we assume that the slot length is at least double the average communication time across the network (i.e., $\Delta \geq 2\theta$). In turn, the action of attester $A_{(i,n)}$ is constrained by $\nu_{(i,n)} = 1 \implies \tau_{(i,n)} \geq t_n + \delta_{n,(i,n)}$.

[1] Note that we have a continuum of attesters, rather than a discrete set. In Ethereum, over 19,000 attesters emit a vote per slot (as of 2023-06-16).

2.1 Block proposers

The pay-off for proposer n is given as follows:

$$U^P(t_n, \phi_n) = \begin{cases} \alpha + \mu \cdot (t_n - t_{n_-})^+ \text{ if } \chi_n = 1 \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

where $\alpha, \mu > 0$ are exogenous constants while t_{n_-} corresponds to the time of the most recent canonical block before slot n and $\chi_n \in \{0, 1\}$ corresponds to whether the block in slot n is canonical on the blockchain. We introduce the conditions for a block to become canonical in our model in the following, and delay until Section 3.2 its interpretation with respect to established consensus models.

Note that we assume that the reward of proposer n increases linearly with time relative to the most recent canonical block so long as block n eventually becomes canonical. This assumption reflects that proposer n accrues incremental MEV over time by delaying the release of their block but that they risk being skipped if they delay release for too long. The time of the most recent canonical block, t_{n_-}, is endogenous where slot n_- refers to the most recent canonical slot and is thus given explicitly as follows:

$$n_- = \max\{k \in \mathbb{N} : \chi_k = 1, k \leq n - 1\} \tag{2}$$

For a block to be canonical, we require both that it receives sufficiently many successful attestations and that the subsequent block producer builds on top of it. More formally, letting $\tilde{A}_n$ denote the successful attestations for block n, χ_n is given explicitly as follows:

$$\chi_n = \begin{cases} 1 & \text{if } \phi_{n+1} = 1, \tilde{A}_n \geq \gamma \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where the number of successful attestations for block n is given as the measure of attesters in slot n voting for block n:

$$\tilde{A}_n = |\{i \in [0, 1] : \nu_{(i,n)} = 1\}| \tag{4}$$

2.2 Attesters

Attester (i, n) receives a pay-off if and only if two conditions are met:

- **Correctness:** A vote by attester (i, n) is correct if their vote is consistent with the canonical blockchain. Recall that the vote of attester (i, n) is given by $\nu_{(i,n)}$ and the eventual canonical status of the block is given by χ_n; thus, this condition is equivalent to $\nu_{(i,n)} = \chi_n$.
- **Freshness:** A vote by attester (i, n) is fresh if it was received by proposer $n+1$ soon enough that it could be included in the block in slot $n + 1$ and the block in slot $n + 1$ is eventually made canonical. We let $\delta_{(i,n),n+1} \sim exp(\theta^{-1})$ denote the random communication time between attester (i, n) and proposer $n + 1$, implying that the first part of this condition equates with $\tau_{(i,n)} + \delta_{(i,n),n+1} \leq t_{n+1}$. Moreover, the second part of this condition equates with $\chi_{n+1} = 1$.

For exposition, we normalize the pay-off for attester (i, n) to unity, implying that their pay-off function is given explicitly as follows:

$$U^A(\nu_{(i,n)}, \tau_{(i,n)}) = \begin{cases} 1 & \text{if } \nu_{(i,n)} = \chi_n, \tau_{(i,n)} + \delta_{(i,n),n+1} \leq t_{n+1}, \chi_{n+1} = 1 \\ \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

3 Analysis

3.1 Equilibrium analysis

There exists a multiplicity of Nash equilibria. In particular, attesters can coordinate to implement proposers acting at any particular time $\Delta^\star \in [0, \Delta]$ within the slot. Formally, we have the following result:

▶ **Proposition 1** (Multiple Equilibria). *For any $\Delta^\star \in [0, \Delta]$, there exists an equilibrium as follows:*

Proposer n selects t_n as follows:

$$t_n = n \cdot \Delta + \Delta^\star \tag{6}$$

and selects ϕ_n as follows:

$$\phi_n = \begin{cases} 1 & \text{if } t_{n-1} \leq (n-1) \cdot \Delta + \Delta^\star \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Attester (i, n) selects $\nu_{(i,n)}$ as follows:

$$\nu_{(i,n)} = \begin{cases} 1 & \text{if (6) and (7) hold} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

and selects $\tau_{(i,n)}$ as follows:

$$\tau_{(i,n)} = \begin{cases} t_n + \delta_{n,(i,n)} & \text{if (6) and (7) hold} \\ n \cdot \Delta & \text{otherwise} \end{cases} \tag{9}$$

Proposition 1 arises because a proposer receives a zero pay-off unless their block earns sufficiently many attestations. In turn, if attesters coordinate on voting for a proposer's block only if the proposer releases their block at a particular time, then the proposer earns a strictly positive pay-off only if she releases their block at that particular time. Thus, since a proposer prefers a strictly positive pay-off to a zero pay-off, each proposer optimally releases their block at the release time on which attesters coordinate.

As an aside, we emphasize that the referenced coordination by attesters is equilibrium behavior. In particular, an attester receives a strictly positive pay-off only if their attestation is correct, and their attestation is correct only if it agrees with the majority of attesters in their slot. As a consequence, when all other attesters vote in one direction, each attester optimally votes in that same direction to avoid a zero pay-off.

Proof. We begin by establishing that (8) - (9) are optimal responses for any attester (i, n). Formally, we take as given that all attesters other than (i, n) follow the equilibrium actions (8) - (9) and also that all proposers follow the equilibrium actions (6) - (7); in that context, we demonstrate that (8) - (9) maximize (5) and thus these are equilibrium actions for each attester (i, n).

If (6) and (7) hold, then $\phi_n = 1$ follows directly for all $n \in \mathbb{N}$. Moreover, if all attesters other than (i, n) follow (8), then (6) and (7) imply $\nu_{(-i,n)} = 1$ which implies $\tilde{A}_n = 1$ for all $n \in \mathbb{N}$. Then, since (6) and (7) imply $\phi_n = 1$ for all $n \in \mathbb{N}$ and also $\tilde{A}_n = 1 \geq \gamma$, (3) therefore implies $\chi_n = 1$ for all $n \in \mathbb{N}$. In turn, since $\nu_{(i,n)} \neq \chi_n$ implies the lowest possible pay-off in (5), we have that $\nu_{(i,n)} = \chi_n = 1$ whenever (6) and (7) holds. If (6) and (7) do not hold, then (8) implies $\nu_{(-i,n)} = 0$ which implies $\tilde{A}_n = 0$ for all $n \in \mathbb{N}$. Moreover, (3) implies $\chi_n = 0$ for all $n \in \mathbb{N}$. In turn, since $\nu_{(i,n)} \neq \chi_n$ implies the lowest possible pay-off, we have that $\nu_{(i,n)} = \chi_n = 0$ whenever the conjunction of (6) and (7) do not hold. Thus, $\nu_{(i,n)} = 1$ is an optimal response if (6) and (7) and $\nu_{(i,n)} = 0$ is an optimal response otherwise, thereby establishing (8) as the equilibrium action for any attester (i, n).

To establish (9) as an optimal response for attester (i, n), note that (5) pointwise decreases in $\tau_{i,n}$ and thus it is optimal to set $\tau_{(i,n)}$ as low as possible subject to feasibility. In general, $\tau_{(i,n)} \geq n \cdot \Delta$ but $\nu_{(i,n)} = 1 \implies \tau_{(i,n)} \geq t_n + \delta_{n,(i,n)} = n \cdot \Delta + \Delta^\star + \delta_{n,(i,n)} > n \cdot \Delta$. As such, whenever $\nu_{(i,n)} = 0$, then $\tau_{(i,n)} = n \cdot \Delta$, whereas whenever $\nu_{(i,n)} = 1$, then $\tau_{(i,n)} = t_n + \delta_{n,(i,n)}$. Then, as per our proof of (8), (6) and (7) imply $\nu_{(i,n)} = 1$ which implies $\tau_{(i,n)} = t_n + \delta_{n,(i,n)}$, whereas if either (6) or (7) does not hold, then $\nu_{(i,n)} = 0$ which implies $\tau_{(i,n)} = n \cdot \Delta$, which thereby establishes (9).

We conclude by demonstrating that (6) - (7) are an optimal response for any proposer n. More formally, we take as given that all attesters follow the equilibrium actions (8) - (9) and also that all proposers other than proposer n follow the equilibrium actions (6) - (7); in this context, we establish that (6) - (7) maximize (5) and thus these are equilibrium actions for each proposer n.

Due to (8), any deviation in (6) or (7) implies $\nu_{(i,n)} = 0$ for all (i, n) which further implies $\tilde{A}_n = 0$. Then, under such a deviation, (3) implies $\chi_n = 0$ which implies a zero pay-off as per (1). Finally, since pay-offs are bounded below by zero, not deviating from (6) and (7) necessarily produces a higher pay-off than any such deviation and thus (6) and (7) are equilibrium actions. ◀

3.2 Model justification

The model presented in Section 2 is an idealized description of a blockchain consensus mechanism. A sequence of proposers is selected, each of which is given the right to produce a block for the slot they are assigned to in the sequence. Once the block is released, a set of attesters assigned for the current slot gets to vote for the presence or absence of the block.

When the proposer chooses to build on the previous block, they affirm its place in the canonical chain. There is no block tree: either the current proposer recognizes the block produced by the proposer before them as part of the canonical chain ($\phi = 1$), or they recognize that the previous proposer failed to produce a block which is part of the canonical chain ($\phi = 0$). With the assumption of a continuum of attesters, at equilibrium, sufficiently many votes reach the following proposer, allowing them to make the call on whether or not the previous proposer's block is canonical.

This model resembles the Streamlet protocol [13]. A proposer submits a block for consideration to the rest of the network. If $\gamma = 2/3$ share of attesters vote the block in, the block is notarized. If attesters do not, e.g., because the block is unavailable, the chain height is not increased, but the next slot starts, giving the opportunity to the next block producer to submit a block for consideration. Leaders extend the longest chain of notarized blocks they have seen.

The model also bears resemblance with the proposed (block, slot) fork choice rule of the Ethereum Gasper protocol [1], specifically the dynamically available chain produced by the protocol, when $\gamma = 1/2$. In this model of the fork choice, attesters submit a vote attesting to the presence or absence of a block at some given slot. The canonicity of a block is however complicated by the LMD-GHOST rule for block weight accumulation. Obtaining more than half of the attesters' vote may then neither be a sufficient nor a necessary condition to be part of the canonical chain.

Generally, we formulate the hypothesis that most Proof-of-Stake-based leader selection protocols will be exposed to timing games. As long as duties are assigned according to an absolute (wall-clock) time schedule, there exists no pressure to complete duties in a timely manner comparable to the random arrival process of leaders in Proof-of-Work. For instance, PBFT-based finalization protocols such as Tendermint [21] or HotStuff [31] do not perform a view change until some timeout is reached, which a leader may use to time their release appropriately. While a sufficiently decentralized committee of validators is an existing feature of these protocols, our model further highlights its role in enforcing timeliness at equilibrium, as described in Section 3.1.

4 An empirical case study: Ethereum

Following a formal analysis of the coordination game between proposers and attesters, we now investigate the occurrence of such strategic timing games in real-world systems. To this end, we examine Ethereum, an ideal candidate for the empirical analysis of potential timing games, owing to its mature MEV market structure and the availability of accessible, informative data points.

We show that timing games are indeed worth playing. However, we find that proposers do not delay their block release with the intention to capture more MEV. Instead, we find that delays are mostly due to latency in their signing processes. Thus, we can conclude that timing games are rational to engage in, but do not yet occur to their full possible extent.

4.1 Consensus mechanism

The Ethereum consensus mechanism is a composite of two protocols: variants of LMD GHOST [29] and Casper FFG [9], often referred to together as Gasper [10]. In this paper, we focus exclusively on Ethereum's *available chain* that is built roughly following LMD GHOST. This is because timing games only occur on the available chain. Within this protocol, time progresses in 12-second slots [3]. For each slot, one consensus participant, referred to as a validator, is selected as the *block proposer*. According to the honest validator specifications [4], which define the rules for honest protocol participation, a block should be released at the beginning of the slot (0 seconds into the slot). Furthermore, the protocol selects a committee of *attesters* from the validator set who vote on what they consider to be the latest canonical block as soon as they hear a valid block for their assigned slot, or 4 seconds into the slot, whichever comes first [4]. We refer to this 4-second mark as the *attestation deadline*. This dynamic, in which block proposers must release their block early enough for attesters to receive it via the peer-to-peer network before the attestation deadline, results from the attestation deadline serving as a coordination Schelling point [27]. It is worth noting that the honest validator specification prescribes block proposers to release their block at the beginning of the slot, while attesters only attest 4 seconds into the slot (unless a valid block is heard prior to the attestation deadline). This opens up room for block proposers to release their block strategically – i.e., as late as possible while ensuring they accumulate a sufficient share of attestations.

4.2 Block production process

To assess the potential benefits of timing games for block proposers, it is important to comprehend the value of time and the process by which MEV opportunities are captured in the block proposing process. In Ethereum, the MEV market structure evolved and matured significantly over time, turning the block production process into an intricate interplay between specialized actors [30]. This division of labor enables validators to profit from MEV without engaging in the complex process of identifying MEV opportunities themselves. Instead, validators can outsource the task of building a maximally profitable block to an out-out-protocol block auction process known as MEV-Boost [5].

Searchers look for MEV opportunities (e.g., arbitrages), and submit bundles of transactions alongside bids to express their order preference to *block builders*. Block builders, in turn, specialize in packing maximally profitable blocks using searcher bundles and other available user transactions before submitting their blocks with bids to *relays*. Relays act as trust facilitators between block proposers and block builders, validating blocks received by block builders and forwarding only valid headers to validators. This ensures validators cannot steal the content of a block builder's block, but can still commit to proposing this block by signing the respective block header. In the long run, Ethereum's plans include enshrining this currently out-of-protocol mechanism into the protocol [7, 8] to eliminate relay trust assumptions. It is worth noting that MEV-Boost is an opt-in protocol, and validators can always choose to revert to local block building. Finally, when a validator is selected to propose a block in a given slot, they request the highest-bidding block header from the relay, sign it, and return the signed block header to the relay, which then releases the block to the peer-to-peer network.

In summary, searchers find MEV opportunities and express their transaction-ordering preferences within a block via bids. Block builders aim to build maximally profitable blocks using searcher bundles and user transactions, then submit their block content and bids to relays. Validators ultimately request the highest-paying block header, sign it and return it to relays, which release the signed block to the peer-to-peer network. Due to competition at all levels in this block production process (except for the block proposing monopoly), the block proposer is able to capture most of the MEV via this block auction.

MEV-Boost block auction

Here, we granularly outline the sequence of events that take place during the block construction of MEV-Boost block auctions on the Ethereum network. Figure 1 illustrates these events along with their corresponding timestamps, and is intended to serve as a reference for the remainder of this empirical analysis.

The auction for the block of slot n begins in slot $n-1$ (at $t = -12000$ms), during which builders submit blocks alongside bids to relays. This competitive process between block builders determines the right to construct the block for slot n and secures potential MEV-derived profits (block building profit equates to extracted MEV minus bid value). For each bid, the relay logs the timestamps of events at which the bid was received by the relay (`receivedAt`). After some validity checks are completed by the relay, the bid is made available to the proposer (`eligibleAt`). When the proposer chooses to propose a block [2], the proposer requests `getHeader` to receive the highest bidding, eligible block header from the relay. Upon receiving the header associated with the winning bid, the proposer signs it and thereby commits to proposing this block built by the respective builder in slot n. The signed block header is sent to the relay, along with a request to get the full block content from the relay (`getPayload`). Finally, the relay receives the signed block header (`signedAt`)

[2] An honest participant will request the block header shortly before slot n such that the block can be released on time, at the beginning of slot n ($t = 0$ms).

and publishes the full block contents to the peer-to-peer network and proposer. As soon as peers see the new block, validators assigned to the slot can attest to it. This cycle completes one round of consensus repeating every slot.

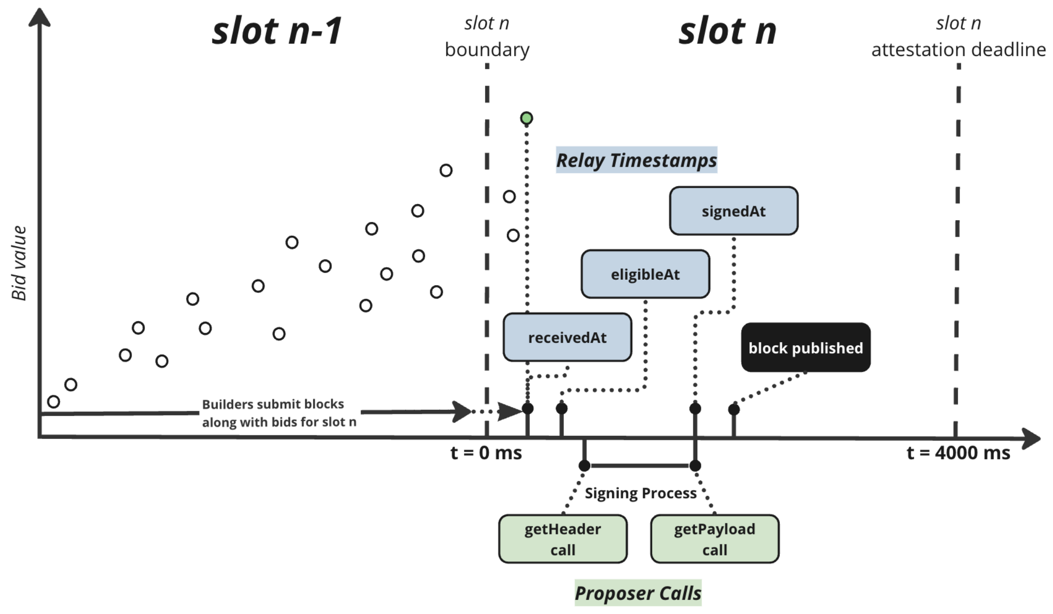


Figure 1 Logical representation of the block production process for slot n. Builder bids begin streaming in during slot $n-1$, after which the proposer and relay interact through requests and responses.

4.3 Data sets

The analysis utilizes data provided by the *ultra sound relay* from March 4, 2023, to April 11, 2023. This covers just under 185,000 slots, interspersed from slot 5,965,398 to slot 6,282,397, and includes all bids placed by block builders through this relay. There were over 800 bids per slot, for a total of over 150 million bids. The winning block originated from the *ultra sound relay* for nearly 85,000 of these slots, and so we measure timestamps and other properties for those slots when investigating winning bids specifically. Finally, we augmented the winning slots with various on-chain measures from the execution layer (EL) and consensus layer (CL), such as attestations and aggregations, using a combination of analytical tools like Dune and direct observation of the peer-to-peer network.

4.4 Are timing games worth playing?

Marginal value of time

Timing games offer potential for substantial profit due to the increased MEV opportunities they provide. First, we assess whether timing games are worth playing for proposers, by estimating the incremental MEV gained per second. We utilize all bids submitted by builders from the *ultra sound relay* to examine the relationship between the timestamp at which the relay received a bid (`receivedAt` timestamp relative to the slot boundary) and the bid value, residualized against slot fixed effects to account for differences between low- and high-MEV regimes and other unobservable forms of heterogeneity. We then fit a regression line to this relationship, obtaining a slope with a coefficient of 0.0065 ETH per second, an estimate for the marginal value of time. Figure 2 depicts the linear increase in median bid values over the slot duration on a point-by-point basis, and the distribution of bid receival times, indicating that most bids are submitted between four seconds before the slot boundary to one second after. This analysis shows there exists a linear positive marginal value of time, indicating that a rational block proposer would participate in timing games.

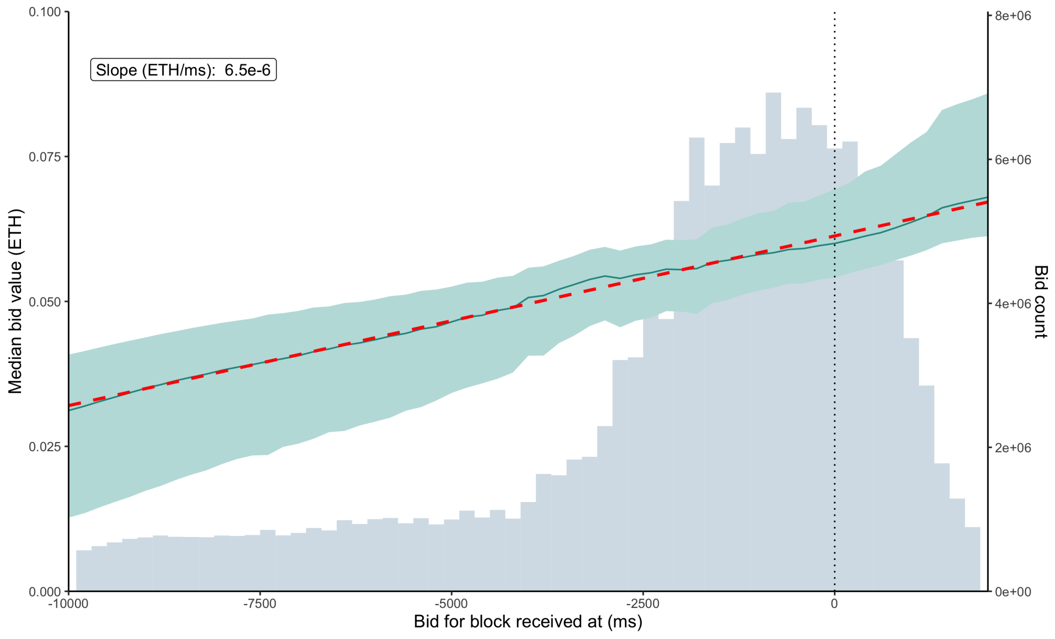


Figure 2 Analysis of bid values and their distribution over slot duration. The histogram (in blue) shows the distribution of bid counts across time. The dark green line represents the median bid value in Ether (ETH) for each time bin (with its associated IQR in green), residualized against the slot fixed effects that are estimated in a linear regression of bid on timestamp (dashed red line). The x-axis shows time in milliseconds relative to the slot boundary, the left y-axis displays the residualized bid value in ETH, and the right y-axis displays the count of bids.

4.5 Are block proposers playing timing games?

Having shown that timing games are worth playing, we turn our attention to whether proposers are currently taking advantage of the opportunity to accumulate more MEV by committing to a bid later than foreseen by the honest validator specifications.

Characterizing late block signing behavior

First, we investigate whether block proposers sign headers and associated bids later than the slot boundary ($t = 0$), the time stipulated by the honest protocol specifications to broadcast their block to the network. We observe that meadian winning bid is signed 774 ms after the slot boundary ($t_{(111573)} = 575.5$, $p < 1 \times 10^{-20}$, using a two-tailed paired Student t-test) and about 513 ms after the relay made the bid eligible ($t_{(111573)} = 472.6$, $p < 1 \times 10^{-20}$, using a two-tailed paired Student t-test). Figure 3a displays the distribution of timings for winning bids, based on *ultra sound relay* timestamps for bid reception from the builder (`receivedAt`, median = 157ms), eligibility for proposer signing (`eligibleAt`, median = 260ms), and the actual signing by proposers (`signedAt`, median = 774ms). To better understand the reasons behind late-signing behavior by proposers, we map validator public keys to their staking entities and CL clients, see Figure 3b and 3c respectively). Validator to staking entity mappings were obtained via a combination of open source data sets [3], and validator to client mappings were obtained using blockprint [2], an open source tool assigning client labels to validators based on their attestation packing on the Ethereum beacon chain. We found that staking entities such as Kraken ($t = 38.9$, $p < 1 \times 10^{-20}$) and Coinbase ($t = 67.6$, $p < 1 \times 10^{-20}$), as well as proposers using the Lodestar client ($t = 44.9$, $p < 1 \times 10^{-20}$) sign block headers significantly later than other block proposer types (results were obtained using two-tailed unpaired Student t-tests). Notably, additional analysis is required to differentiate the interdependencies between validator entities and clients. This analysis shows that proposers are signing blocks significantly later than expected, but it does not yet clarify the underlying reasons, which could include participation in timing games or increased latency for independent reasons, e.g., longer signing processes.

We subsequently collected data for 1241 slots (slot 6,200,251 to 6,204,957 on April 11, 2023) and used the time difference between `getHeader` and `getPayload` calls by proposers as an approximation to estimate the duration of the signing process (see Figure 1). Figure 4 shows that the median difference between `getHeader` and `getPayload` call is 418 ms. Interestingly, this delay, attributable to the signing process, accounts for 75.42% of the overall latency. This percentage was determined by calculating the difference between the signing time and the moment the bid was deemed eligible by the relay on a slot-by-slot basis, using the formula $\frac{\text{median}(\texttt{getPayload}-\texttt{getHeader})}{\text{median}(\texttt{signedAt}-\texttt{eligibleAt})} \times 100$. We conclude that late signing behavior is primarily attributed to latency caused by the signing process, rather than intentional delays to incorporate more MEV in blocks. This finding aligns with the hypothesis that large US-based staking entities, such as Coinbase and Kraken, may prefer utilizing sophisticated remote secure signing mechanisms, resulting in a lengthier signing process compared to other parties.

4.6 The impact of latency on the peer-to-peer network

Our prior results indicate that validators are largely not engaging in timing games to accrue more MEV. Nonetheless, we assess the implications of late signing of consensus messages on the peer-to-peer network. Specifically, we examine the relationship between relay timestamps and the time at which (1) blocks and (2) attestations and aggregations are first seen by the rest of the peer-to-peer network. The consensus layer data was obtained through nodes run by the Ethereum Foundation, for 2643 slots (slots 6,357,601 to 6,363,807 on May 3 and 4, 2023). Figure 5a shows the sequence of these event timestamps over the course of a slot. We subsequently assess the correlations between each of these event pairs, as depicted in Figure 5b. Our analysis reveals high correlations between the time at which blocks are signed by proposers (i.e., the `signedAt` relay timestamp) and the time at which blocks (correlation coefficient = 0.986) and attestations (correlation coefficient = 0.971) are initially observed by the peer-to-peer network. These findings underscore the significance of proposers signing blocks promptly, as it considerably impacts the downstream processes at the consensus layer in the network.

[3] Dune Spellbooks: `https://dune.com/spellbook`, Mevboost.pics Open Data: `https://mevboost.pics/data.html`

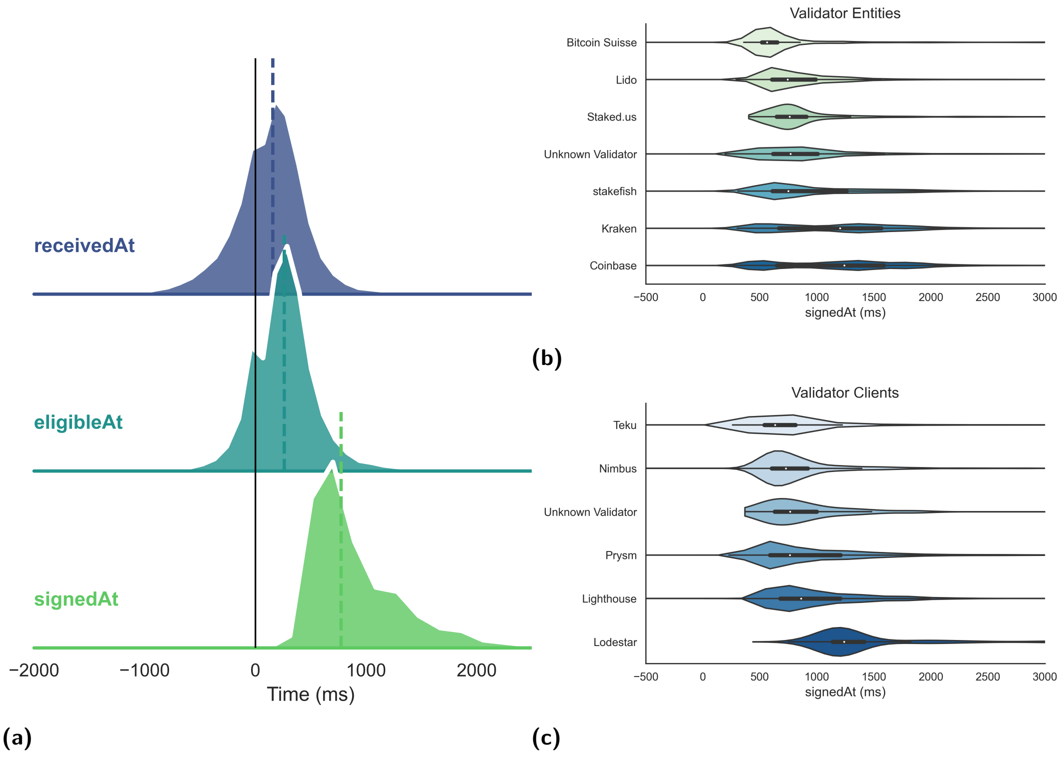
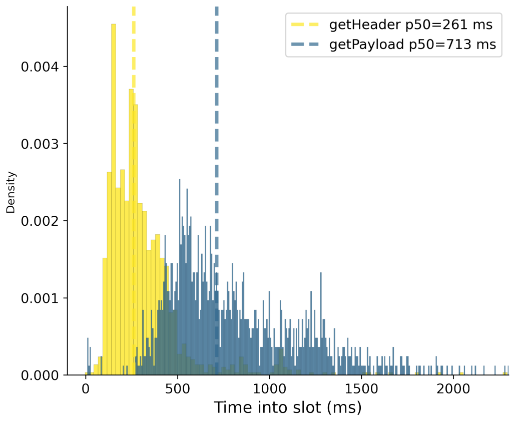


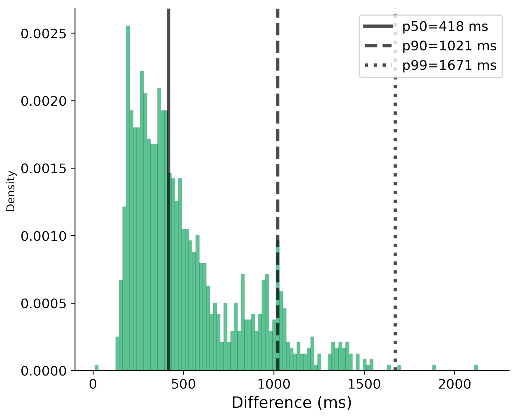
Figure 3 Analysis of event timestamps and their distributions among Validator Clients and Entities. (3a) Multiple Kernel Density Estimation (KDE) distributions of event timestamps from the relay data, showing the probability density functions for three event types: `receivedAt` (blue), `eligibleAt` (green), and `signedAt` (light green). (3b-3c) Violin plots comparing the distribution of `signedAt` event timestamps for the top 7 validator entities and clients. The x-axis represents time in milliseconds (ms) relative to the slot boundary, while the y-axis displays validator clients and entities, ordered by the mean `signedAt` time. The width of each violin plot signifies the kernel density estimation of the `signedAt` event timestamps, demonstrating the distribution and frequency of the events within each group.

We evaluate the impact of the release time of block n on the share of attestations that vote for block n and are included on-chain in block $n+1$. As a reminder, attestations of slot n are only included on-chain one or more slots later (slot $\geq n+1$). We use the time at which blocks are signed by proposers (`signedAt`) as a heuristic for the block release time. In our analysis, we focus on attestations included in the subsequent slot and compute a metric *next-slot attestation share*. This metric refers to the share of attestations voting for block n that are included in block $n+1$, out of the total number of slot n attestations included in block $n+1$ that vote for any block other than n. The argument is that if a proposer releases their block n too late, attesters will not receive it in time to vote for it before their attestation deadline ($t = 4000$ms, see Figure 1, and [4]). Hence, in such scenarios attesters vote for a different block (e.g., the parent block). This is captured by the next-slot shares metric.

Figure 6 shows that late blocks do indeed cause a steep drop-off in the share of attestations voting for that block that are included in the subsequent block. We observe that the next-slot attestation share remains close to one for as long as the block is signed within the first two seconds of the slot. Once the two second threshold is crossed, there is a substantial



(a)



(b)

Figure 4 Estimating the latency induced by the signing process. (4a) Histogram of `getHeader` and `getPayload` call timestamps relative to slot boundary. The histogram displays the density of events occurring at different times into the slot (in milliseconds) for `getHeader` (yellow) and `getPayload` (blue) calls. (4b) Histogram of the time difference between `getHeader` and `getPayload` calls. The histogram shows the density of time differences (in milliseconds) `getHeader` and `getPayload` calls. Vertical lines represent the 50th (solid), 90th (dashed), and 99th (dotted) percentiles of the distribution.

drop-off and many winning blocks earn fewer than half of the next-slot attestation share, which continues to rapidly decrease towards zero as we approach the theoretical $t = 4000$ms attestation deadline. These results demonstrate the impact of untimely block proposals on the rest of the peer-to-peer network and highlight the importance of signing and broadcasting blocks on time in order to prevent fair rewards for honest consensus participation, missed slots and chain reorganizations.

5 Discussion

In this paper, we present an argument that consensus participants are subject to exogenous incentives, primarily MEV, that exist outside the consensus mechanism itself. This highlights the imperative for blockchain protocols to ensure economic fairness among all consensus participants. Specifically, it necessitates a design where honest and rational consensus participation become indistinguishable, and honesty within the protocol is the most profitable strategy. This approach ensures that rational participants have no incentive to deviate from honest consensus participation.

We present a model that highlights the time-dependent value for consensus participants and probes into the strategic timing considerations that block proposers face. Our model uncovers a spectrum of equilibria wherein attesters can enforce any deadline for block proposals to achieve canonical status, thereby emphasizing the crucial role of Schelling points as coordination mechanisms. For instance, in the Ethereum network we observe the emergence of such Schelling points through the default settings of client software. The widespread use of these default settings among consensus participants generally ensures their effectiveness.

We support our theoretical findings by observations of the Ethereum network. Our analysis demonstrates that timing games are indeed worth playing for block proposers, enabling them to capture additional MEV by delaying their block proposals beyond the

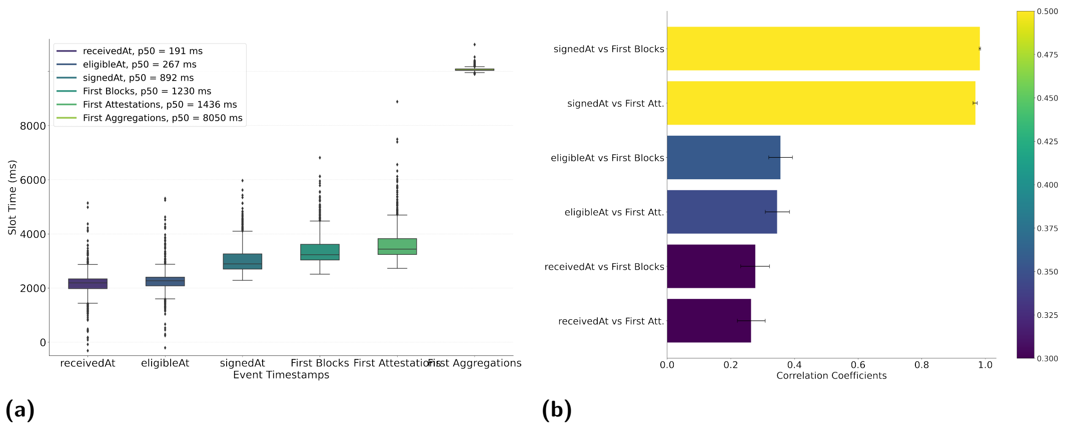


Figure 5 Analysis of relay and consensus layer timestamps. (5a) Box plot of the time differences between relay and consensus timestamps. The box plots display the distribution of time differences for `receivedAt`, `eligibleAt`, and `signedAt` events, as well as blocks, attestations, and aggregations first seen by the peer-to-peer network. The boxes represent the interquartile range (IQR) from the first quartile (Q1, 25th percentile) to the third quartile (Q3, 75th percentile), while the whiskers extend to the minimum and maximum values within three times the IQR. The horizontal lines within the boxes represent the median values. (5b) Bar plot of Pearson correlation coefficients for each pair of event timestamps. The bars represent the mean correlation coefficient for each relationship, while the error bars represent the 95% confidence intervals obtained via bootstrapping.

timeframe prescribed by the honest validator specification. However, we observe that current instances of delayed block proposals are primarily due to latency in the block signing process, rather than a conscious strategy to maximize profits. The apparent lack of maximal MEV capture by honest proposers could be attributed to either a lack of common knowledge or existing social norms around this practice. It's clear, however, that these are not sustainable safeguards for maintaining economic fairness.

The implications of timing games are manifold and significant. An rational participant who engages in timing games will outperform honest participants, leading to a centralization of stake over time. Hypothetically, this could culminate in a breach of consensus security. In a more practical sense, it may encourage individual stakers to delegate their stake to professional entities adept at these practices, negatively impacting the network's decentralization. Moreover, timing games can overload the messaging system within a short time span, potentially causing cascading failures at the peer-to-peer layer, particularly within client systems.

Essentially, timing games are facilitated by the monopolistic right that block proposers possess for a single round of consensus. Introducing competition in block proposing, similar in effect to the exogenous randomness in Proof of Work (PoW) systems, emerges as a potential solution. However, the challenge lies in deterministically selecting a winning proposer, or reverting to peer-to-peer latency races, which in itself is centralizing. Alternatively, an on-chain heuristic for timely block proposals could incentivize timely participation, yet the allure of MEV rewards might still outweigh any in-protocol consensus rewards. Tackling the root cause of timing games remains an open challenge.

In the Ethereum context, a late-block reorging mechanism has been adopted in the fork choice, effectively imposing a 4-second deadline for block proposers. This constraint significantly limits the extent to which block delays are possible. Looking ahead, the adoption of (block, slot) type of attestations is likely, further refining the protocol. However, it

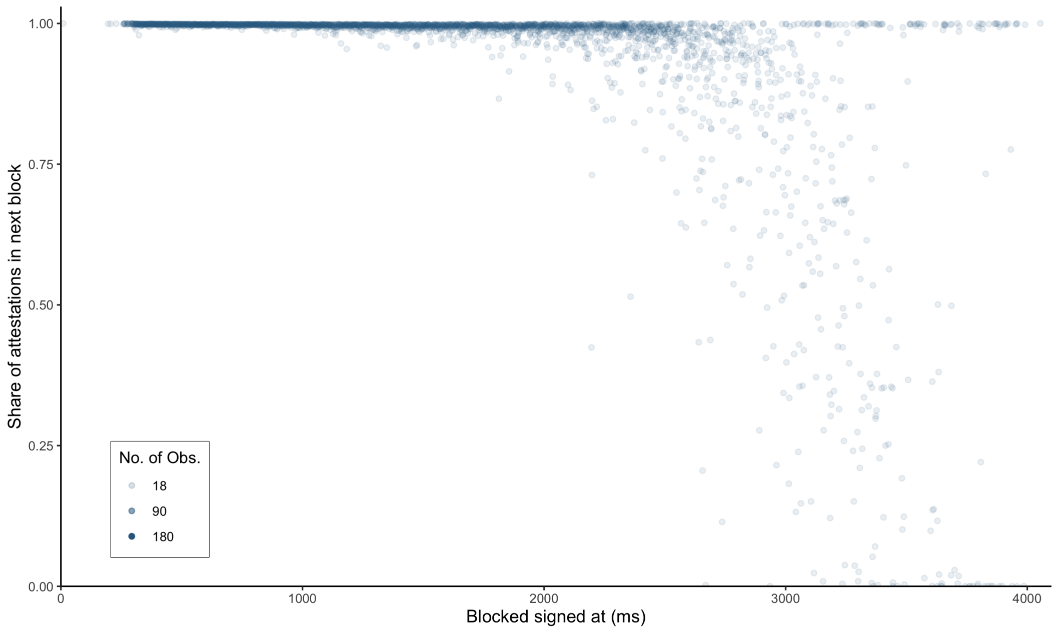


Figure 6 Effect of block release time on share of attestations that vote for said block and are included in subsequent block. The `signedAt` timestamps are used as a heuristic for release time. Each point on the graph corresponds to the time (in milliseconds) at which the winning block was signed within the slot and the average share of attestations voting for it that are included in the next block.

remains challenging to address the root cause of timing games, as it is deeply intertwined with the fundamental workings of Proof of Stake (PoS). Although limiting the length of the proposer's interval is feasible, completely eliminating the monopolistic market structure of block proposers proves to be a difficult task.

Consequently, it may prove valuable to find a more general abstraction for PoS type of protocols and further explore the implications of consensus participants being exposed to incentives outside of consensus itself, such as MEV. More generally, assuming rational as opposed to honest type of consensus participation should prove significant in designing economically fair blockchain protocols. Revisiting the existing literature on consensus mechanisms through the lens of rational as opposed to honest consensus participation should prove valuable.

References

1 (block, slot) fork choice. `https://github.com/ethereum/consensus-specs/pull/2197`. Accessed: 2023-10-05.

2 blockprint. Accessed: 2023-10-05. URL: `https://github.com/sigp/blockprint`.

3 Ethereum consensus specifications - beacon chain. Accessed: 2023-10-05. URL: `https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md`.

4 Ethereum consensus specifications - honest validator. Accessed: 2023-10-05. URL: `https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md`.

5 Mev-boost. Accessed: 2023-10-05. URL: `https://github.com/flashbots/mev-boost`.

6 Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. Clockwork finance: Automated analysis of economic security in smart contracts. *arXiv preprint arXiv:2109.04347*, 2021.

7 Vitalik Buterin. Proposer/block builder separation-friendly fee market designs. Accessed: 2023-10-05. URL: https://ethresear.ch/t/proposer-block-builder-separation-friendly-fee-market-designs/9725.

8 Vitalik Buterin. Two-slot proposer/builder separation. Accessed: 2023-10-05. URL: https://ethresear.ch/t/two-slot-proposer-builder-separation/10980.

9 Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv:1710.09437 [cs.CR]*, 2019. URL: https://arxiv.org/abs/1710.09437.

10 Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining ghost and casper. *arXiv preprint arXiv:2003.03052*, 2020.

11 Christian Cachin and Marko Vukolić. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

12 Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 acm sigsac conference on computer and communications security*, pages 154–167, 2016.

13 Benjamin Y Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 1–11, 2020.

14 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *CoRR*, abs/1904.05234, 2019. arXiv:1904.05234.

15 Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. No more attacks on proof-of-stake ethereum? *arXiv preprint arXiv:2209.03255*, 2022.

16 Francesco D'Amato and Luca Zanolini. Recent latest message driven ghost: Balancing dynamic availability with asynchrony resilience. *arXiv preprint arXiv:2302.11326*, 2023.

17 Francesco D'Amato and Luca Zanolini. A simple single slot finality protocol for ethereum. *arXiv preprint arXiv:2302.12745*, 2023.

18 Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and nakamoto always wins. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 859–878, 2020.

19 Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.

20 Kose John, Thomas Rivera, and Fahad Saleh. Economic implications of scaling blockchains: Why the consensus protocol matters. *NYU Stern Working Paper*, 2023.

21 Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11), 2014.

22 Kevin Liao and Jonathan Katz. Incentivizing blockchain forks via whale transactions. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*, pages 264–279. Springer, 2017.

23 Barnabé Monnot. Timing games in proof-of-stake. Accessed: 2023-10-05. URL: https://ethresear.ch/t/timing-games-in-proof-of-stake/13980.

24 Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 446–465. IEEE, 2021.

25 Joachim Neu, Ertem Nusret Tas, and David Tse. Two more attacks on proof-of-stake ghost/ethereum. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, pages 43–52, 2022.

26 Fahad Saleh. Blockchain without waste: Proof-of-stake. *Review of Financial Studies*, 34(3):1156–1190, 2021.

27 Thomas C Schelling. *The Strategy of Conflict: with a new Preface by the Author*. Harvard university press, 1980.

28 Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 560–576. Springer, 2022.

29 Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in Bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

30 Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: Mev countermeasures: Theory and practice, 2022. doi:10.48550/ARXIV.2212.05111.

31 Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

Practical Large-Scale Proof-Of-Stake Asynchronous Total-Order Broadcast

Orestis Alpos
University of Bern, Switzerland

Christian Cachin
University of Bern, Switzerland

Simon Holmgaard Kamp
Aarhus University, Denmark

Jesper Buus Nielsen
Aarhus University, Denmark

Abstract

We present simple and practical protocols for generating randomness as used by asynchronous total-order broadcast. The protocols are secure in a proof-of-stake setting with *dynamically changing* stake. They can be plugged into existing protocols for asynchronous total-order broadcast and will turn these into asynchronous total-order broadcast with dynamic stake. Our contribution relies on two important techniques. The paper "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using Cryptography" [Cachin, Kursawe, and Shoup, PODC 2000] has influenced the design of practical total-order broadcast through its use of threshold cryptography. However, it needs a setup protocol to be efficient. In a proof-of-stake setting with dynamic stake this setup would have to be continually recomputed, making the protocol impractical. The work "Asynchronous Byzantine Agreement with Subquadratic Communication" [Blum, Katz, Liu-Zhang, and Loss, TCC 2020] showed how to use an initial setup for broadcast to asymptotically efficiently generate sub-sequent setups. The protocol, however, resorted to fully homomorphic encryption and was therefore not practically efficient. We adopt their approach to the proof-of-stake setting with dynamic stake, apply it to the Constantinople paper, and remove the need for fully homomorphic encryption. This results in simple and practical proof-of-stake protocols.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Total-Order Broadcast, Atomic Broadcast, Proof of Stake, Random Beacon

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.31

Related Version *Full Version*: https://eprint.iacr.org/2023/1103

Funding This work has received funding from the Swiss National Science Foundation (SNSF) under grant agreement Nr. 200021_188443 (Advanced Consensus Protocols). Simon Holmgaard Kamp and Jesper Buus Nielsen are partially funded by The Concordium Foundation.

1 Introduction

State of the art. It is well known that Asynchronous Total-Order Broadcast (ATOB) cannot be deterministic [25]. The necessary randomness is usually modelled as a *common coin* scheme [33], informally defined as a source random values observable by all participants but unpredictable for the adversary [10]. Common coins are most practically implemented using threshold cryptography [11, 23, 32, 10]. This approach has many benefits. It is conceptually simple and efficient, it achieves optimal resilience $t < n/3$, where n the number of parties running the protocol, and it results in a *perfect coin*, meaning that it is uniformly distributed and agreed-upon with probability 1. The drawback, however, is that it requires a trusted setup or an Asynchronous Distributed Key Generation (ADKG) protocol. Current state of the art ADKG protocols [20, 1, 2] have communication cost of $O(\lambda n^3)$, where λ is the security parameter.

Given that state-of-the-art ATOB protocols have communication complexity $O(\lambda n^2)$, or even amortized $O(\lambda n)$, it is evident that the communication cost of ADKG becomes the bottleneck. In a permissioned setting with a static set of parties, it is common to proactively refresh the threshold setup [9]. In a Proof-of-Stake (PoS) setting, particularly, where the stake is constantly evolving and parties may dynamically join or leave the protocol, the ADKG protocol must be run periodically. Recent literature on asynchronous consensus uses committees, which contain only a subset of the parties, reducing the communication complexity of BA even further to $O(\lambda n \log n)$ at the cost of tolerating only $t < (1-\epsilon)n/3$ corruptions for any $\epsilon > 0$ [5, 18]. As the protocol run by the committee assumes an honest supermajority, this paradigm comes with one of two significant drawbacks. Either the sampled committee has to be very large, so that its maximal corruption remains below $n/3$ with overwhelming probability [22]. Otherwise, in order to keep the committee size small, the corruption level in the ground population must be assumed lower than $n/3$ by a considerable margin. Directly porting this idea to ADKG results in the same drawbacks. Finally, existing DKG protocols support only flat structures, where every party has the same weight and in total $t < n/3$ parties are corrupted. They do not readily work for a setting where every party holds a different share of the stake.

Seeds in PoS protocols. PoS-based ATOB protocols and blockchains require, apart from common coins, a second type of randomness, usually referred to as a *seed.* In PoS blockchains there is the notion of *accounts* with stake on them, of *roles*, such as "produce the 42-nd block", and of a *lottery*, through which accounts win the right to execute roles. This is typically [21, 26] implemented using a *Verifiable Pseudo-Random Function* (VRF) [30]: each account has a private key for a VRF and applies it to the role, producing a pseudorandom value. If this value is above a threshold then the account wins the right to execute the role. However, for this approach to work the lottery needs as input not only a role but also a seed. Without the seed, a party can operate with several accounts and move all its stake to the luckiest account. By including a seed in the lottery and using the stake distribution from a point in time when the seed was unpredictable one can mitigate this attack [21].

In practice one can use a common-coin protocol to produce the seeds. We remark, however, that the two randomness-generation protocols have different requirements. A common-coin scheme does not have to be always unpredictable and agreed-upon, but only with some constant probability [31, 12]. It should, however, be efficient, as it is used in every agreement instance within the broadcast protocol. On the other hand, the seed-generation protocol must always be unpredictable and agreed upon, but it can be slow, as it is only run periodically (e.g., once per epoch).

Related work. Multiple common-coin constructions without a trusted dealer have been proposed in the literature. Ben-Or [4] presents a simple protocol, where every party flips a *local coin.* As a result, parties agree on the value of the coin only with probability $\Theta(2^{-n})$, A common-coin scheme from *verifiable secret sharing* has been shown by Canetti and Rabin [12], but their resulting Byzantine agreement protocol has communication complexity $\mathcal{O}(n^{11})$. Patra, Choudhury, and Rangan [31] bring this down to $\mathcal{O}(n^3)$.

A different approach constructs common coins from *publicly verifiable secret sharing.* The resulting protocols [13, 14, 19, 36, 38] are efficient, yet they all make synchrony assumptions. RandShare [38] has been formalized in the asynchronous communication setting but it is not scalable, as it requires $\mathcal{O}(n^3)$ communication per party. Another line of work is based on *time-based cryptography.* Protocols in this category [29, 16] employ *verifiable delay functions* [7] and rely on the assumption that certain functions (such as exponentiation in groups of unknown order [35]) can only be computed serially. None of the aforementioned works explicitly mentions the network assumptions. Overviews of random beacon protocols are given by Raikwar and Gligoroski [34], and by Choi, Manoj, and Bonneau [17].

Multiple works that circumvent ADKG exist in the literature, but they either make more assumptions, have non-optimal resilience, or result in inefficient protocols. Existing PoS blockchains rely on the timely delivery of honestly generated blocks, hence make timing assumptions. Ouroboros Praos [21] implements a randomness beacon protocol, used as seed in their leader-election algorithm, by hashing a large number of VRF outputs. Partial-synchrony assumptions assure that the honestly generated VRF outputs cannot be delayed arbitrarily by the adversary. King and Saia [27, 28] propose a synchronous common-coin protocol that makes uses of pseudorandomly selected committees, but achieves non-optimal resilience. This is improved in the protocol of Algorand [26, 15], where each committee member applies a VRF on the seed of previous block, and then the smallest valid VRF value sent by some committee member is kept. The protocol is first described in the synchronous model [26] and later extended to the partially synchronous [15]. Cohen, Keidar, and Spiegelman [18] extend this idea to the asynchronous model, but their protocol achieves an $n = 4.5t$ resilience. In all these protocols the coins are not reusable and the whole coin-generation algorithm has to be run repeatedly.

Blum *et al.* [5] also generate randomness without ADKG. Their ATOB protocol works in the following way. Assume first that a trusted dealer publishes on a ledger all the setup material required for one instance of Byzantine agreement and one instance of a Multiparty Computation (MPC) protocol. Then, on every invocation of the agreement protocol, parties use the Byzantine-agreement setup in the agreement protocol and the MPC setup in a tailor-made MPC protocol that refreshes the whole setup. Finally, they replace the trusted dealer with a standard MPC protocol, executed once in a distributed setup phase. This blueprint solves the problem of dynamic stake elegantly, but, the proposed MPC protocol for refreshing the setup, which has to be executed for *every* Byzantine agreement instance, is not efficient: it employs Threshold Fully Homomorphic Encryption (TFHE), digital signatures, and zero-knowledge proofs.

Contributions. In this paper we address all the aforementioned limitations of randomness generation for the first time. We present asynchronous seed-generation and common-coin protocols that

- require no trusted setup,
- support optimal resilience $t < n/3$,
- employ small committees and are concretely efficient,
- directly support the PoS setting and dynamic participation,
- are modular and can be generically used in any ATOB broadcast.

Our methods. We are motivated by the question whether one can use the simple, practical, and efficient approach of getting common coins from threshold setup without running inefficient and complicated protocols whenever the stake has shifted. Building on the idea of Blum *et al.* [5], we rely on the fact that there already exists a functional ATOB: we generate the setup assuming that we already have the ATOB, and then use the generated setup to

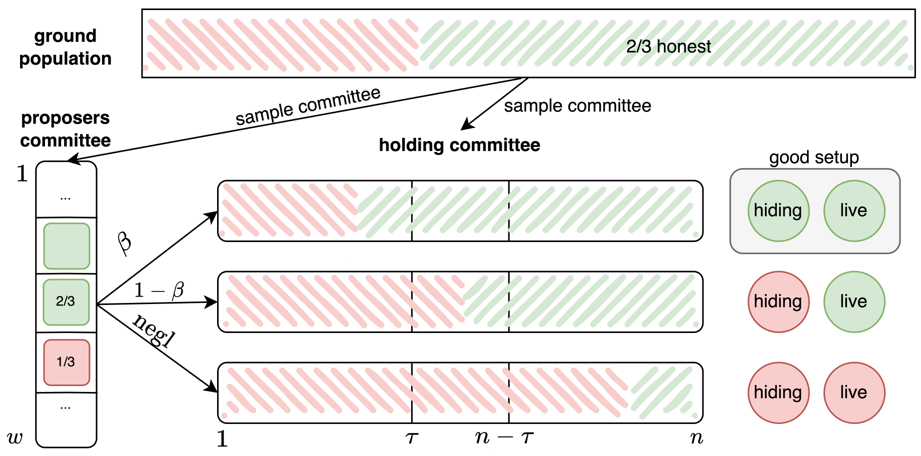


Figure 1 The high level idea of our protocols. A *proposers committee* is elected, and we wait until w proposers broadcast a setup. Assuming $^2/_3$ honesty in the ground population, a proposer is honest with probability $^2/_3$. Each proposer is assigned a *holding committee* of size n and creates an (n, τ) threshold setup for it. A committee is *hiding* if it contains at most τ corrupted parties, and *live* if it contains at most $n - \tau - 1$ corrupted parties. A setup is *good* if its proposer in honest and its holding committee is hiding and live. We set w so as to have enough honest proposers, and n and τ so that each holding committee is hiding with constant probability β and live with all but negligible probability. As a result, we get good setups with a constant probability γ.

keep the ATOB running. To maintain practical efficiency the crucial step is to avoid FHE. We achieve this by generating weaker setups than Blum *et al.* [5], nonetheless still strong enough for the continued execution of the ATOB.

A crucial observation is that coins consumed by Byzantine agreement do not need to be perfect, i.e., always unpredictable and agreed upon [12, 31]. Hence, instead of generating a single, perfect threshold setup, we generate several candidate setups, such that some *constant* fraction of them are good. Many DKG protocols can be seen as doing this as their first step, but their next step is to combine them into a single perfect setup. In order to be combinable, the setups must be of a particular form, and the committee that holds the setup must be *good* (that is, contain less than a threshold corruptions) except with negligible probability. As our setups are not combined and our committees only need to be good with a constant probability, our protocols are simpler and more efficient, and use smaller committees.

Both seed, our seed-generation protocol, and wMDCF, our common-coin protocol, follow the approach depicted in Figure 1. They elect a *proposers committee*, each member of which is expected to create a *setup* (a *VSS setup* or a *coin setup*, for seed and wMDCF, respectively). Each elected proposer is assigned a *holding committee*, for which it creates the threshold setup. For this, the proposer acts as a dealer, encrypts the private setup material under the keys of the holding committee, and broadcasts these encryptions and the required verifications keys with a single message on the ATOB. We use a VRF-based lottery to determine both the proposers and the holding committees, where each party is elected with probability proportional to its stake. To open a seed value in the seed protocol, each of the holding committees reveals its shares and these are all added together. To flip a coin cid in the wMDCF protocol we first open a new seed value and then hash cid with the seed to obtain a pointer to one of the published setups, which is used to obtain the value of cid.

As we show in Section 7, we can have a proposers committee of size 653 and holding committees of size 359, resulting in approx. $85K$ encrypted values posted on Ledger. For 60 bits of security and assuming optimal corruption $^1/_3$ in the ground population, our protocols are live with all but negligible probability, and our common-coin protocol is unpredictable and agreed-upon with probability approx. 31.8%. It is instructive to compare these results against previous literature, particularly against the approach that runs the randomness-generation protocols in committees with honest supermajority. Algorand [26, Figure 3] requires a committee of size approx. 2000, assuming corruption 0.2 in the ground population, and larger than 4000, assuming corruption 0.24, to get good committees with probability $5 \cdot 10^{-9}$, or approx. 28 bits of security. Extending this approach to a ground population with corruption 0.3, which is still sub-optimal, and 60 bits of security, the authors of GearBox [22, Table 1] show that committees of size 16037 are needed. We remark that asynchronous distributed key generation protocols, the state-of-the-art approach for threshold-setup generation, require honest supermajority, hence one would require a committee of similar sizes and sub-optimal resilience in the ground population.

Organization. The rest of this paper is organized as follows. Section 2.1 presents the formal model used in the schemes and Section 2.2 presents the primitives used in our schemes. Then, each of the seed-generation and common-coin protocols are presented in modular way, in two steps. Section 3 presents *wVSS*, a weak verifiable secret sharing scheme, which is then used in Section 4 to build the *seed-generation* protocol. Section 5 presents wHDCF, a weak honest-dealer coin-flip protocol, which is then used in Section 6 to build the wMDCF common-coin protocol. All of these schemes are parameterized over committee sizes and thresholds, and secure bounds for these are computed in Section 7.

2 Preliminaries

2.1 Model

We assume a model with asynchronous authenticated point-to-point channels. In addition we assume an asynchronous persistent total-order broadcast channel. We denote by Ledger the totally-ordered sequence of messages that have been delivered on the channel. We point out that if a blockchain has a distinction between final and non-final messages, then Ledger denotes the final messages. We assume that when a protocol is started all the parties taking part in the protocol agree on a session identifier sid and an existing point on the ledger, $p \leq |\mathsf{Ledger}|$. We think of p as the *starting point of the protocol*, which gives consensus on the context of the protocol like stake distribution and lottery as discussed below. Protocols can have *public output* which might not be explicitly posted on the ledger, but will have a well-defined value and virtual point p at which they happened.

▶ **Definition 1** (Public output). *We say that* PubOutF *is a public output function if it computes a public output from a ledger* Ledger *and a session identifier* sid*, where either* $\mathsf{PubOutF}(\mathsf{Ledger}, \mathsf{sid}) = y \in \{0,1\}^*$ *or* $\mathsf{PubOutF}(\mathsf{Ledger}, \mathsf{sid}) = \perp$*. We require that if* $\mathsf{PubOutF}(\mathsf{Ledger}, \mathsf{sid}) \neq \perp$ *then* $\mathsf{PubOutF}(\mathsf{Ledger}\|m, \mathsf{sid}) = \mathsf{PubOutF}(\mathsf{Ledger}, \mathsf{sid})$ *for all* m*. We say that* sid *gave public output* y *at position* p *if* $|\mathsf{Ledger}| \geq p$ *and* $\mathsf{PubOutF}(\mathsf{Ledger}[1, p-1], \mathsf{sid}) = \perp$ *and* $\mathsf{PubOutF}(\mathsf{Ledger}[1,p], \mathsf{sid}) = y$*. Unless multiple* sid*'s are in scope, we omit the* sid *parameter. Finally, we informally say that some protocol gives public output* PubOutF *when additionally the ledger is implicit or when it is an eventual property of the ledger.*

Dynamic Stake. We consider proof-of-stake defined via the ledger. For each Ledger there is a stake distribution $\Sigma(\mathsf{Ledger}) : \mathbb{P} \to \mathbb{R}_0$ which may change as the ledger grows, can be computed in poly-time, and which gives for each party P its stake $\Sigma(\mathsf{Ledger})(\mathsf{P})$. For each point p there is also a stake distribution Σ_p, which is the stake distribution used by protocols with p as starting point. It may be different from $\Sigma(\mathsf{Ledger}[1, p])$, as discussed below.

Lotteries. In PoS based protocol it is common that parties are selected at random for carrying out a role in the protocol, like serving on a committee or producing the next block in a blockchain. To keep the model simple we assume that this is done via a random oracle. To keep the model simple we assume that for each point p on the ledger there is a random oracle $\Gamma_p : \{0,1\}^* \to \{0,1\}^\lambda$. We assume that Γ_p is sampled and made available to the parties at some point *after* Σ_p can be computed from Ledger. This ensures that Γ_p is independent of Σ_p. If Γ_p was made available before Σ_p was fixed, then corrupted parties would be able to update Σ_p based on Γ_p (e.g., by moving their stake to parties "lucky" in Γ_p). We implement this by iteratively generating random and unpredictable seeds, appearing as public outputs, with our seed protocol. Then, for a given point p, let seed_p be the latest seed on $\mathsf{Ledger}[1,p]$, let $\Gamma_p(x) = R(\mathsf{seed}_p, x)$, for a random oracle R, let $p' < p$ be the latest point where seed_p was unpredictable, and let $\Sigma_p = \Sigma(\mathsf{Ledger}[1,p'])$.

Our protocols include steps where a party samples a committee cid of size n. We model this as a function $\mathsf{SampleCommittee}_p(\mathsf{cid}, n) \to \left(\mathsf{H}_i\right)_{i\in[n]}$ that uses Γ_p to sample (with replacement) n parties from $\mathbb{P}$ with probability proportional to the stake Σ_p. As the input is public, the output can be verified by a function $\mathsf{VerifyCommittee}_p(\mathsf{cid}, \left(\mathsf{H}_i\right)_{i\in[n]})$ that reruns $\mathsf{SampleCommittee}_p(\mathsf{cid}, n)$ and verifies that it matches $\left(\mathsf{H}_i\right)_{i\in[n]}$. We assume $\mathsf{SampleCommittee}_p(\mathsf{cid}, n)$ is locally computable by every party. Using our lottery abstraction this could be implemented by calling $\Gamma_p(\mathsf{cid}, i)$, for some committee cid and for $i \in [n]$, to obtain a number $r_i \in \{0, 2^\lambda - 1\}$, and then deterministically mapping r_i to a party $P_i \in \mathbb{P}$ based on Σ_p. Observe that a party with relatively large stake can appear multiple times in the committee.

2.2 Primitives

Our schemes make use of the following primitives.

2.2.1 Public-Key Encryption with Full Decryption

There are keys $(\mathsf{dk}_i, \mathsf{ek}_i)$, for all $\mathsf{P}_i \in \mathbb{P}$, for an IND-CPA encryption scheme with full decryption, PKE. Encrypting a message $m \in \mathsf{PKE}.\mathcal{M}$ using randomness $r \in \mathsf{PKE}.\mathcal{R}$ results in a ciphertext $c = \mathsf{Enc}_{\mathsf{ek}_i}(m; r) \in \mathsf{PKE}.\mathcal{C}$. Given a ciphertext $c \in \mathsf{PKE}.\mathcal{C}$ the decryption algorithm $\mathsf{Dec}_{\mathsf{dk}_i}(c)$ returns both $m \in \mathsf{PKE}.\mathcal{M}$ and $r \in \mathsf{PKE}.\mathcal{R}$. The triple (m, r, c) can then be verified by anyone holding ek_i by checking if $\mathsf{Enc}_{\mathsf{ek}_i}(m; r) = c$. Given an invalid ciphertext a zero knowledge proof that the ciphertext is invalid can be obtained using the secret key.

Construction using El Gamal. We first show that we can obtain the properties above in the random oracle model, as long as only encryptions of random messages are needed. This can then be lifted to a complete encryption scheme by symmetrically encrypting the message under a freshly sampled random key.

To encrypt a random value r, use El Gamal with $H(r)$ as randomness. I.e. if $\mathsf{dk} = x$ and $\mathsf{ek} = h = g^x$, then you encrypt r as $c = (A, B) = (g^{H(r)}, r \cdot h^{H(r)})$. To decrypt you first compute $r = B/(A^x)$, then check if re-encrypting using $H(r)$ as randomness gives back c. If verification checks out you can simply send r as proof. If the re-encryption does not match, you provide a proof that r was obtained by decrypting c. Note that (A, B) decrypts to r (under (g, h)) iff $\mathrm{DL}_g(h) = \mathrm{DL}_A(B/r)$, so this proof can be constructed using the Fiat-Shamir transform of the Σ-protocol for equality of discrete logarithms.

In the full scheme, in order to encrypt m using randomness r, you encrypt r as above and additionally include a symmetric encryption of m using r as key. To decrypt you first use regular El Gamal decryption to obtain r and verify it by re-encrypting. If it was encrypted correctly you use it to decrypt m and return (m, r), otherwise return $(\perp, r)$.

2.2.2 Threshold Coin Flip

We use a (n, t)-threshold coin-flip (CF) scheme, where n is the total number of parties, t is the corruption threshold, and the reconstruction threshold is $t + 1$. The scheme has the following interface.

- $\mathsf{Setup}(n, t) \rightarrow (\mathsf{vk}, \mathsf{sk}_1, \dots, \mathsf{sk}_n)$: The dealer generates a verification key vk and secret key shares sk_i of P_i. The secret keys can be used to create coin shares of multiple coins.
- $\mathsf{VerifyKeyShare}(\mathsf{vk}, i, \mathsf{sk}_i) \rightarrow b \in \{0, 1\}$: Given the verification keys vk, it verifies sk_i.
- $\mathsf{Flip}(\mathsf{sk}_i, \mathsf{coin}) \rightarrow (\mathsf{s}_i, \mathsf{w}_i)$: Given a coin identifier coin and secret key sk_i, it returns a coin share s_i for coin and potentially a correctness proof w_i, i.e., a proof that the coin share has been computed correctly using sk_i.
- $\mathsf{VerifyCoinShare}(\mathsf{vk}, \mathsf{coin}, \mathsf{s}_i, \mathsf{w}_i) \rightarrow b \in \{0, 1\}$: It verifies coin share s_i for coin identifier coin using the correctness proof w_i and verification key vk.
- $\mathsf{Combine}(\mathsf{coin}, \{\mathsf{s}_{i_j}\}_{j \in [t+1]}) \rightarrow \mathsf{s} \in \{0, 1\}^\lambda$: Given $t + 1$ valid coin shares s_{i_j}, for $j \in [t + 1]$, it returns the value s of the coin identifier coin.
- $\mathsf{VerifyCoin}(\mathsf{vk}, \mathsf{coin}, \mathsf{s}) \rightarrow b \in \{0, 1\}$: It verifies s as the value of coin identifier coin using the verification key vk.

Security properties. Assuming an honest dealer, i.e., that $\mathsf{Setup}()$ is correctly executed, and that there are no more than t corrupted parties, the scheme satisfies the following.

Completeness If the dealer is honest then all key shares generated with $\mathsf{Flip}(\mathsf{sk}_i, \mathsf{coin})$ will verify with VerifyCoinShare.

Agreement For any $t + 1$ valid key shares the value $\mathsf{Combine}(\mathsf{coin}, \{\mathsf{s}_{i_j}\}_{j \in [t]})$ is the same, which define *the* value $\mathsf{s}_{\mathsf{coin}}$.

Unpredictability The value $\mathsf{s}_{\mathsf{coin}}$ is unpredictable without honest shares, i.e., for a set $C = \{\mathsf{P}_{i_j}\}_{j \in [t+1]}$ of corrupted parties, if a poly-time adversary has been given vk and sk_i for $\mathsf{P}_i \in C$ for a random setup and has not been given $\mathsf{Flip}(\mathsf{sk}_i, \mathsf{coin})$ for $\mathsf{P}_i \notin C$, then it cannot guess $\mathsf{s}_{\mathsf{coin}}$ better than at random. This holds even if it has access to an oracle giving $\mathsf{Flip}(\mathsf{sk}_i, \mathsf{coin}')$ for all honest P_i for all $\mathsf{coin}' \neq \mathsf{coin}$.

Instantiation. Scheme CF can be instantiated with any non-interactive unique threshold signature scheme, such as BLS threshold signatures [8, 6]. The dealer picks a random secret key sk and shares it among all n parties using a polynomial $\phi(X) = \sum_{k=0}^{t} \phi_k X^k$, such that $\phi_0 = \mathsf{sk}$. The only difference from threshold BLS is in $\mathsf{Setup}()$: it runs the key generation algorithm of the threshold signature scheme, but it does not return the verification keys in the form $g_2^{\mathsf{sk}_i} \in G_2$, where $i \in [n]$ and g_2 is the generator of G_2, as in the original scheme. Instead, it returns a vector $(V_0, \dots, V_t)$, where $V_k = g^{\phi_k} \in G_2$, for $k \in \{0, \dots, t\}$, i.e., it returns Feldman commitments [24] to the coefficients of ϕ. This allows us to implement $\mathsf{VerifyKeyShare}()$, so P_i can verify that its key share sk_i is indeed a point on polynomial ϕ by checking whether

$$g_2^{\mathsf{sk}_i} \overset{?}{=} \prod_{k=0}^{t} (V_k)^{i^k}. \tag{1}$$

Observe that the original verification keys can still be obtained using (1) with input vk and i, hence $\mathsf{VerifyCoinShare}()$ and $\mathsf{VerifyCoin}()$ need no modification. Algorithm $\mathsf{Flip}()$ returns a signature share s_i on message coin using the key share sk_i of party P_i. Algorithm $\mathsf{Combine}()$ creates the threshold signature s from $t + 1$ valid signature shares, which can then be hashed to get a value in $\{0,1\}$. Algorithms $\mathsf{VerifyCoinShare}()$ and $\mathsf{VerifyCoin}()$ invoke the signature verification algorithm, which, in the case of BLS, only takes as input the message coin and a signatures share s_i or signature s, i.e., $\mathsf{w}_i = \bot$, and uses a pairing function. Alternatively, one can use the common-coin scheme of Cachin, Kursawe, and Shoup [10], but $\mathsf{VerifyCoin}()$ would additionally need as input the $t+1$ valid coin shares and proofs $\{\mathsf{s}_{i_j}, \mathsf{w}_{i_j}\}_{j\in[t+1]}$.

2.2.3 Secret sharing

We require a secret-sharing scheme TSS with threshold t with the following interface.

1. $\mathsf{Share}(s;r) \to (s_1,\dots,s_n)$: It shares a secret s using randomness r to n secret shares $(s_1,\dots,s_n)$.
2. $\mathsf{Reconstruct}(\{s_{i_j}\}_{j=1}^{t}) \to s'$: Given t shares it reconstructs some secret s'.

The hiding property says that the joint distribution of t shares s_i is independent of s. We can instantiate TSS with Shamir's secret sharing scheme [37].

2.2.4 Digital Signature

Finally, there are keys $(\mathsf{sk}_\mathsf{P}, \mathsf{vk}_\mathsf{P})$, for all $\mathsf{P} \in \mathbb{P}$, for a digital signature scheme DS with unique signatures.

3 Weak Verifiable Secret Sharing

In this section we define a weak VSS protocol. It is weak in the sense that it is sometimes not hiding. But it is always binding and live (allows reconstruction). There is a designated dealer D, which is one of the participating parties. We assume D is given as part of session identifier, $\mathsf{sid} = (\mathsf{D}, \mathsf{sid}')$, and hence is known by all parties when the instance is created.

Syntax. The syntax of wVSS is as follows.

Commit On input $(\textsc{commit}, \mathsf{sid}, m)$ to D it starts running the commitment protocol and may as a result help produce a public output (see Definition 1) $\mathsf{PubOutVSSCommit}$. On input $(\textsc{commit}, \mathsf{sid})$ to a participating party $\mathsf{P} \neq \mathsf{D}$ it starts running the commitment protocol and may as a result help produce a public output $\mathsf{PubOutVSSCommit}$.

Open On input $(\textsc{open}, \mathsf{sid})$ after $\mathsf{PubOutVSSCommit}(\mathsf{sid}, \mathsf{Ledger}_\mathsf{P}) \neq \bot$, a party P starts running the open protocol and may as a result output $(\textsc{done-open}, \mathsf{sid}, m_\mathsf{P}, \pi)$, where π is a proof that m_P is the output. The proof can be checked by any party P' for which $\mathsf{PubOutVSSCommit}(\mathsf{sid}, \mathsf{Ledger}_{\mathsf{P}'}) \neq \bot$ using $\mathsf{wVSSVerify}(\pi, m)$.

Security. The security properties of wVSS are as follows.

Termination

(1) If D is honest and gets input $(\textsc{commit}, \mathsf{sid}, m)$, and all other honest parties get input $(\textsc{commit}, \mathsf{sid})$ then eventually there is a public output $\mathsf{PubOutVSSCommit}$.

(2) If $\mathsf{PubOutVSSCommit}$ occurred and all honest parties get input $(\textsc{open}, \mathsf{sid})$ then eventually all honest parties give an output $(\textsc{done-open}, \mathsf{sid}, \cdot)$.

(3) If any honest party gives an output $(\textsc{done-open}, \mathsf{sid}, \cdot)$ then eventually all honest parties give output $(\textsc{done-open}, \mathsf{sid}, \cdot)$.

Validity If D is honest and had input $(\textsc{commit}, \mathsf{sid}, m)$ and some honest P gave output $(\textsc{done-open}, \mathsf{sid}, m_\mathsf{P}, \pi)$, then $m_\mathsf{P} = m$ and $\forall m' : \mathsf{wVSSVerify}(\pi, m') = \top \Leftrightarrow m' = m$.

Binding If D is corrupted, then the following holds. When the first honest party observes public output PubOutVSSCommit then one can in poly-time compute from the view of the adversary up to this point, a message m such that if later an honest party P gives output $(\textsc{done-open}, \mathsf{sid}, m_{\mathsf{P}}, \pi)$ then $m_{\mathsf{P}} = m$ and $\forall m' : \mathsf{wVSSVerify}(\pi, m') = \top \Leftrightarrow m' = m$.

β-Weak Hiding If D is honest then for each session sid it holds with probability $\beta > 0$ at a point in time t before any honest party got input $(\textsc{open}, \mathsf{sid})$ that m is hidden in the view of the adversary at time t, i.e., if $m = m_b$ for (m_0, m_1) picked by the adversary and a uniformly random bit b, then the adversary cannot guess b better than at random.

Construction. The central idea of our wVSS construction is to have a dealer choose a secret seed σ and secret share it unto a random holding committee and put on the ledger an encryption of each of the shares under the public key of the holder, this vector of encryptions is called the setup. If this was done correctly any $t+1$ honest parties can decrypt their shares and use them to reconstruct σ. All randomness for the secret sharing and the encryption will be generated from σ using a PRG. This allows the committee to rerun the setup procedure and check consistency with the published setup after reconstruction. This ensures that if anyone $t+1$ parties can reconstruct to some value σ, then all shares are correct, and therefore all subsets of $t+1$ shares reconstruct to the same σ. We also use randomness derived from σ to encrypt m and include the encryption in the setup. We cannot let the dealer pick the holding committee as we need enough honest parties on it to avoid deadlock of reconstruction. Therefore the holding committee is sampled pseudorandomly from the session identifier sid.

We use n_{VSS} and τ_{VSS} to define the size of the holding committee sampled by the dealer, and the reconstruction threshold in the holding committee, respectively. To ensure weak hiding these parameters should be chosen such that the sampled committee has at most τ_{VSS} corruptions with constant probability at least β, and to ensure liveness less than $n_{\text{VSS}} - \tau_{\text{VSS}}$ should be corrupted except with negligible probability. The scheme makes use of a signature scheme DS (Section 2.2.4), an encryption scheme PKE with full decryption (Section 2.2.1), a threshold secret-sharing scheme TSS (Section 2.2.3), a pseudorandom generator PRG, and a hash function H modelled as a random oracle.

Algorithm 1 Scheme wVSS, algorithm Commit, where an instance sid of wVSS is created at point p on Ledger. Code for process P_i.

```
1: function commit_value($\sigma, \mathcal{C}$)
2:     $\rho = \mathsf{PRG}(H(\sigma))$
3:     $m_{mask} \overset{\$\rho}{\leftarrow} \{0,1\}^{\lambda}$
4:     $(s_1, \ldots, s_{n_{\text{VSS}}}) \overset{\$\rho}{\leftarrow} \mathsf{TSS.Share}(\sigma)$
5:     for $j$ in $\mathcal{C}$ do
6:         $r_j \overset{\$\rho}{\leftarrow} \{0,1\}^{\lambda}$;   $e_j \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}_j}(s_j, r_j)$
7:     return $((e_1, \ldots, e_{n_{\text{VSS}}}), m_{mask})$

8: upon input $(\textsc{commit}, \mathsf{sid}, \pi, m)$ where $\mathsf{sid} = (\mathsf{D}, \mathsf{sid}')$ and $\mathsf{P}_i = \mathsf{D}$ do        // only dealer D
9:     $(\mathsf{H}_1, \ldots, \mathsf{H}_{n_{\text{VSS}}}) \leftarrow \mathsf{SampleCommittee}_p(\mathsf{sid}, n_{\text{VSS}})$
10:    $\sigma \leftarrow \mathsf{DS.Sign}_{\mathsf{sk}_\mathsf{D}}(\mathsf{sid})$
11:    $((e_1, \ldots, e_{n_{\text{VSS}}}), m_{mask}) \leftarrow \mathsf{commit\_value}(\sigma, (\mathsf{H}_1, \ldots, \mathsf{H}_{n_{\text{VSS}}}))$
12:    broadcast $(\mathsf{sid}, \pi, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m \oplus m_{mask})$ on Ledger
```

We implement Commit in Algorithm 1. In order to commit to a chosen value m, D first pseudorandomly samples a holding committee of size n_{VSS} (line 9). We say that the committee is "assigned" to D, as D cannot influence it without getting rejected as public output. The dealer then computes a signature σ on sid, obtaining a unique and unpredictable value. Then D commits to σ by secret-sharing it to the committee. This logic is extracted in an auxiliary function commit_value. It computes a random tape $\rho = \mathsf{PRG}(H(\sigma))$. This random tape is used in all subsequent steps that require randomness. Specifically, in line 3 a random message m_{mask} is sampled, in line 4 the value σ is secret-shared to the members of the holding committee $\mathcal{C}$ using an $(n_{\text{VSS}}, \tau_{\text{VSS}})$-TSS, and in lines 5– 6 the shares of σ are encrypted to the committee members. Each of these values are sampled *pairwise independently* from ρ. Finally, D broadcasts its *VSS setup* on Ledger (line 12). This VSS setup serves as a public output signalling that the message is committed and can at this point only be opened to some unique value–which could be $\perp$. We define the function $\mathsf{PubOutVSSCommit}(\mathsf{Ledger}, (\mathsf{D}, \mathsf{sid}'))$ as the earliest (in Ledger) message $\big((\mathsf{D}, \mathsf{sid}'), \pi, (e_1, \mathsf{H}_1) \dots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m\big)$ which is signed by D, where $\mathsf{VerifyCommittee}((\mathsf{D}, \mathsf{sid}'), \mathsf{H}_1, \dots, \mathsf{H}_{n_{\text{VSS}}}) = 1$. If no such message exists in Ledger, then $\mathsf{PubOutVSSCommit}(\mathsf{Ledger}, \mathsf{sid}) = \perp$.

Algorithm 2 Scheme wVSS, algorithm Open, where an instance sid of wVSS is created at point p on Ledger. Code only for process P_i is in the committee of instance sid, i.e., P_i is one of the H_j in the VSS-setup $(\mathsf{sid}, (e_1, \mathsf{H}_1) \dots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m_{masked})$ published on Ledger.

```
    State:
13:     validShares[sid] ← [ ]

14: upon input (OPEN, sid) such that $\mathsf{PubOutVSSCommit}(\mathsf{Ledger}_{\mathsf{P}_i}, \mathsf{sid}) \neq \perp$ do
15:     let $((e_1, \mathsf{H}_1) \dots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m_{masked}) = \mathsf{PubOutVSSCommit}(\mathsf{Ledger}_{\mathsf{P}_i}, \mathsf{sid})$
16:     let $\mathcal{C} = \{\mathsf{H}_1, \dots, \mathsf{H}_{n_{\text{VSS}}}\}$
17:     $(s'_i, r'_i) \leftarrow \mathsf{Dec}_{\mathsf{dk}_i}(e_i)$
18:     $e'_i \leftarrow \mathsf{Enc}_{\mathsf{ek}_i}(s'_i, r'_i)$
19:     if $e'_i = e_i$ then
20:         send (SHARE, $s'_i, r'_i$) to parties in $\mathcal{C}$
21:     else
22:         create zk-proof $\mathsf{W}_i$ that $e_i$ decrypts to $(s'_i, r'_i)$
23:         send (COMPLAINTENCRYPTION, sid, $\mathsf{W}_i, s'_i, r'_i$) to parties in $\mathcal{C}$

24: upon deliver (SHARE, $s_j, r_j$) from $\mathsf{P}_j$ do
25:     if $e_j = \mathsf{Enc}_{\mathsf{ek}_j}(s_j, r_j)$ then
26:         append $s_j$ to validShares[sid]

27: upon |validShares[sid]| = $\tau_{\text{VSS}} + 1$ do
28:     let $\big(s_{j_1}, \dots, s_{j_{\tau_{\text{VSS}}+1}}\big)$ = validShares[sid]
29:     $\sigma' \leftarrow \mathsf{TSS.Reconstruct}(\{s_{j_k}\}_{j \in [\tau_{\text{VSS}}+1]})$
30:     if $\mathsf{DS.Ver}_{\mathsf{vk_D}}(\sigma', \mathsf{sid}) = 0$ then
31:         output (DONE-OPEN, sid, $\perp$, validShares[sid])
32:     $((e'_1, \dots, e'_{n_{\text{VSS}}}), m_{mask}) \leftarrow \mathsf{commit\_value}(\sigma')$
33:     if $(e'_1, \dots, e'_{n_{\text{VSS}}}) \neq (e_1, \dots, e_{n_{\text{VSS}}})$ then
34:         output (DONE-OPEN, sid, $\perp$, validShares[sid])
35:     else
36:         output (DONE-OPEN, sid, $m_{mask} \oplus m_{masked}, \sigma'$)

37: upon deliver $c$ = (COMPLAINTENCRYPTION, sid, $\mathsf{W}_j, s_j, r_j$) do
38:     $e'_j \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}_j}((s_j, r_j))$
39:     if $e'_j \neq e_j$ and $\mathsf{W}_j$ is valid then
40:         output (DONE-OPEN, sid, $\perp$, $c$)
```

We implement Open in Algorithm 2. On input OPEN, and after $\mathsf{PubOutVSSCommit}(\mathsf{Ledger}, \mathsf{sid}) \neq \bot$, a party in the holding committee of the sid instance parses the output as a VSS setup. Party P_i then decrypts e_i to get its share and the randomness used for encryption, (s_i, r_i) (line 17). It then re-encrypts (s_i, r_i) (line 18) to verify that the encryption was done correctly (line 19). If the encryption is valid they send (s_i, r_i) to the other parties, otherwise it sends a verifiable complaint (lines 22–23). The complaint includes a that e_i decrypts to (s'_i, r'_i).

Upon receiving a share from P_j (line 24), party P_i verifies that the share sent by P_j indeed corresponds to the value e_j published on Ledger. If this is the case, the share is considered valid. Observe that the share the dealer created for P_j might be wrong in the first place. This is detected upon reconstruction. Specifically, once $\tau_{\text{VSS}} + 1$ valid shares are received (line 27), P_i runs the reconstruction of TSS to get back some σ', which should be a signature on sid computed by D. Party P_i first verifies the signature and, if valid, it repeats the steps performed by D to secret-share σ' (line 32). Observe that, given σ', all steps in commit_value() are deterministic. Hence, if the reconstructed σ' is the same as the σ computed by the dealer, then commit_value(σ') will return the same values as the ones posted on Ledger by D or we detect that the dealer cheated and output $\bot$. This is checked in line 33. Finally, upon delivering a complaint, sent by some party P_j, party P_i verifies the complaint and, if valid, outputs $\bot$. Note that no party can produce a valid complaint if the check in line 33 goes through. The wVSSVerify check can simply be implemented by a function that when given an encryption complaint checks if it is valid as in line 37, and when given a set of shares checks that they are all valid and treats them as input to the activation rule in line 27 to see that the same output is obtained. When the output of wVSS needs to be distributed to the full set of parties, each party on the committee simply forwards their $(\text{DONE-OPEN}, \mathsf{sid}, m_{\mathsf{P}}, \pi)$ message to the remaining parties. Note that even though the proof of the outputs can differ, an outside party only needs to receive one. Hence, in gossiping networks the output messages can be deduplicated by only forwarding the first valid one to lower communication complexity.

4 Generating an Unpredictable Seed

In this section we define a seed-generation protocol seed. A seed can be thought of as a perfect coin flip: there is agreement on the output and its value is unpredictable before the protocol starts.

Syntax. The syntax of seed is as follows:

Commit On input (SEED, sid) in a session with session identifier sid a party starts running the commit protocol and may as a result public output PubOutSeedCommit.

Open On input (SEED-OPEN, sid), which must be given after public output PubOutSeedCommit, in a session with id sid a party starts running the opening protocol and may as a result output (DONE-SEED, sid, c), for $c \in \{0, 1\}^\lambda$.

Security. The security properties of seed are as follows:

Termination If all honest parties get inputs (SEED, sid) then eventually all honest parties get public output PubOutSeedCommit.

If all honest parties get correct inputs (SEED-OPEN, sid) then eventually all honest parties give an output (DONE-SEED, sid, $\cdot$).

Agreement If two honest parties have outputs $(\textsc{done-seed}, \mathsf{sid}, c_{\mathsf{P}})$ and $(\textsc{done-seed}, \mathsf{sid}, c_{\mathsf{Q}})$ then $c_{\mathsf{P}} = c_{\mathsf{Q}}$. Call the common value c_{sid}.

Unpredictability For each session sid it holds that c_{sid} is unpredictable before the first honest party gets input $(\textsc{seed-open}, \mathsf{sid})$.

Algorithm 3 Scheme seed, algorithm Commit, where an instance sid of seed is created at some point p on Ledger. Code for process P_i.

```
41: upon input $(\textsc{seed}, \mathsf{sid})$ do
42:     $\mathcal{C} \leftarrow \mathsf{SampleCommittee}(\mathsf{sid}, m_{\textsc{seed}})$
43:     for $j \in [m_{\textsc{seed}}]$ such that $\mathcal{C}[j] = P_i$ do
44:         $r \xleftarrow{\$} \{0,1\}^{\lambda}$
45:         $\mathsf{wVSS}(\textsc{commit}, ((\mathsf{P}_i, j), \mathsf{sid}), r)$
```

Algorithm 4 Scheme seed, algorithm Open, where an instance sid of seed is created at some point p on Ledger. Code for process P_i.

```
46: State:
47:     $\mathsf{openings}[\mathsf{sid}] \leftarrow [\,]$

48: upon input $(\textsc{seed-open}, \mathsf{sid})$ such that $\mathsf{PubOutSeedCommit}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$ do
49:     $\mathsf{setups} \leftarrow \mathsf{PubOutSeedCommit}(\mathsf{sid}, \mathsf{Ledger})$
50:     for $j \in [w_{\textsc{seed}}]$ do
51:         $\mathsf{sid}_j \leftarrow \mathsf{setups}[j]$
52:         $\mathsf{wVSS}(\textsc{open}, \mathsf{sid}_j)$

53: upon deliver $(\textsc{done-open}, \mathsf{sid}_j, r, \pi)$ do
54:     if $j \in [w_{\textsc{seed}}] \wedge \mathsf{wVSSVerify}(\pi, r)$ then
55:         append $m$ to $\mathsf{openings}[\mathsf{sid}]$

56: upon $|\mathsf{openings}| = w_{\textsc{seed}}$
57:     output $(\textsc{done-seed}, \mathsf{sid}, \bigoplus_{r \in \mathsf{openings}[\mathsf{sid}]} r)$
```

Construction. The protocol uses parameters $m_{\textsc{seed}}$ and $w_{\textsc{seed}}$. The idea is to sample $m_{\textsc{seed}}$ parties in $\mathbb{P}$ to contribute a wVSS setup, asynchronously wait for the first $w_{\textsc{seed}}$ setups and use the XOR of them as a seed. We discuss in Section 7 how to set these parameters, such that at least one good setup (that is, from an honest proposer and with a committee with at most τ_{VSS} corrupted members) appears on the ledger, except with negligible probability.

The protocol is started at some starting point p of Ledger, with associated stake Σ_p and committee sampling mechanism $\mathsf{SampleCommittee}_p()$. We implement Commit in Algorithm 3 and Open in Algorithm 4. A party that is elected to contribute a wVSS setup (line 42) picks a random r and starts an instance of wVSS to share r (lines 44–45). Once $w_{\textsc{seed}}$ wVSS protocols with session identifiers $\mathsf{sid}_j = (\mathsf{P}_j, k, \mathsf{sid})$, where $\mathcal{C}[k] = \mathsf{P}_j$, have given public output PubOutVSSCommit on Ledger, then we define PubOutSeedCommit to be the ordered tuple of the session identifiers of the first $w_{\textsc{seed}}$ such outputs. After this point, the value of the nonce is implicitly defined by the state of the ledger, and on input $(\textsc{seed-open}, \mathsf{sid})$, parties start running the Open algorithm on these $w_{\textsc{seed}}$ instances of wVSS (lines 48–52). By design, the holding committee of each of these instances has enough honest members for wVSS to terminate. The final seed value is defined as the XOR of the values output by each Open (line 57).

5 Weak Honest-Dealer Coin-Flip

In this section we define the weak honest-dealer coin-flip (wHDCF) protocol. In wHDCF there is a designated dealer D, which is one of the participating parties. We assume D is given as part of session identifier, $\mathsf{sid} = (\mathsf{D}, \mathsf{sid}')$, and hence is known by all parties when the instance is created. The scheme is *weak* in the sense that parties may output $\perp$ as the value of the coin, but if two honest parties output a value in $\{0, 1\}$, then it will be the same. It is *honest-dealer* as the coin value becomes predictable for a corrupted D. The scheme makes use of a committee verification mechanism $\mathsf{SampleCommittee}_p()$ proportional to stake at point p (Section 2.1), an encryption scheme with full decryption PKE (Section 2.2.1), and an $(n_{\text{COIN}}, \tau_{\text{COIN}})$-threshold weak coin flip scheme CF (Section 2.2.2). Here n_{COIN} and τ_{COIN} are protocol parameters, for which we choose specific values in Section 7.

Syntax. The syntax of weak honest-dealer coin-flip is as follows:

Deal On input $(\text{DEAL}, \mathsf{sid})$ a participating party starts running the dealing protocol of CF and may as a result produce a public output PubOutSingleDeal.

Flip On input $(\text{FLIP}, \mathsf{sid}, \mathsf{cid})$, for coin identifier cid, after $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$, a party starts running the flip protocol of CF and outputs $(\text{DONE-FLIP}, \mathsf{sid}, \mathsf{cid}, s, \pi)$, where $s \in \{\perp\} \cup \{0, 1\}^\lambda$ and π is a proof that s is the output of the coinflipping protocol. The proof can be checked by any party P' for which $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}_{\mathsf{P}'}) \neq \perp$ using $\mathsf{wHDCFVerify}(\pi, m)$.

Security. The security properties of wHDCF are as follows.

Termination

(1) If D is honest and all honest parties get input $(\text{DEAL}, \mathsf{sid})$, then eventually $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$.

(2) If, after $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$, all honest parties get input $(\text{FLIP}, \mathsf{sid}, \mathsf{cid})$, then eventually all honest parties give output $(\text{DONE-FLIP}, \mathsf{sid}, \mathsf{cid}, \cdot)$, except with negligible probability.

Weak Agreement If two honest parties output $(\text{DONE-FLIP}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{P}}, \pi)$ and $(\text{DONE-FLIP}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{Q}}, \pi)$, such that $c_{\mathsf{P}} \neq \perp$ and $c_{\mathsf{Q}} \neq \perp$, then $c_{\mathsf{P}} = c_{\mathsf{Q}}$, except with negligible probability. The same holds if $c_{\mathsf{Q}} \neq \perp$ and $\mathsf{wHDCFVerify}(\pi, c_{\mathsf{Q}}) \neq \perp$. Moreover, if D is honest, then no honesty party P outputs $c_{\mathsf{P}} = \perp$.

Honest-Dealer β-Unpredictability If dealer D of session sid is honest, then each coin flip cid is independently unpredictable with some constant probability $\beta > 0$, where β is defined when $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$ and is independent of cid.

In the full version of this work [3] we formalize the *Honest-Dealer β-Unpredictability* property and show the proofs.

Construction. In a high level, the scheme works as follows. Dealer D is assigned a *coin-holding committee* of size n_{COIN} and creates a *coin setup* for an $(n_{\text{COIN}}, \tau_{\text{COIN}})$-threshold coin scheme CF for this committee. Termination is achieved by appropriately setting the parameters and from the pseudorandom nature of the committee: if the dealer completes the setup, there are at least $\tau_{\text{COIN}} + 1$ honest parties in the committee, except with negligible probability. The weak agreement property is achieved by *verifiable* complaints against a corrupted dealer. Upon receiving a valid complaint, a party terminates the Flip protocol outputting $\perp$. If, additionally, D is honest, then our protocol guarantees unpredictability with constant probability β, defined as the probability of having at most τ_{COIN} corruptions in the committee, and depending only on n_{COIN} and τ_{COIN}.

Algorithm 5 Scheme wHDCF, algorithm Deal, where an instance sid of wHDCF is created at point p on Ledger. Code for process P_i.

```
58: upon input $(\textsc{deal}, \mathsf{sid})$ where $\mathsf{sid} = (\mathsf{D}, \mathsf{sid}')$ and $\mathsf{P}_i = \mathsf{D}$ do        // only dealer D
59:     $(\mathsf{H}_1, \ldots, \mathsf{H}_{n_{\text{COIN}}}) \leftarrow \mathsf{SampleCommittee}_p(\mathsf{sid}, n_{\text{COIN}})$,
60:     $(\mathsf{vk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\text{COIN}}}) \leftarrow \mathsf{CF.Setup}(n_{\text{COIN}}, \tau_{\text{COIN}})$
61:     for $j \in [n_{\text{COIN}}]$ do
62:         $r_j \xleftarrow{\$} \{0,1\}^\lambda$;   $e_j = \mathsf{PKE.Enc}_{\mathsf{ek}_j}((\mathsf{sk}_j, r_j))$
63:     broadcast $(\mathsf{sid}, \mathsf{vk}, (\mathsf{H}_1, e_1), \ldots, (\mathsf{H}_{n_{\text{COIN}}}, e_{n_{\text{COIN}}}))$ on Ledger
```

In Algorithm 5 we implement Deal. The dealer first (line 59) samples the coin-holding committee of size n_{COIN} and then (line 60) uses CF to create a coin setup for it. The coin setup includes secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\text{COIN}}}$ and verification key vk. Each secret key sk_i is encrypted to party's P_i long term private key ek_i using a fresh randomness r_i (lines 61–62). The coin setup is broadcast on Ledger. When a coin-setup is included in Ledger we define the public output $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger})$ as $(\mathsf{vk}, (\mathsf{H}_1, e_1), \ldots, (\mathsf{H}_{n_{\text{COIN}}}, e_{n_{\text{COIN}}}))$ if the included committee verifies using VerifyCommittee. Otherwise the output is $\perp$.

In Algorithm 6 we implement Flip. Only parties in the coin-holding committee run it. When P_i gets input $(\textsc{flip}, \mathsf{sid}, \mathsf{cid})$ and $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}_{\mathsf{P}_i}) \neq \perp$, it first reads the coin setup and tries to decrypt e_i to obtain its key share (line 70). Scheme PKE returns sk_i' and the randomness r_i' that D is supposed to have used at encryption time. Party P_i checks whether D has indeed done so by re-encrypting (sk_i', r_i') and checking the result against e_i. If it is different, P_i sends a $\textsc{complaintEncryption}$ message that includes a zero-knowledge proof that e_i decrypts to (sk_i', r_i') (lines 71–73) and stops handling the Flip event. Otherwise, P_i can prove correct decryption of e_i in a complaint message by sending (sk_i', r_i'). Party P_i then verifies its key share against the verification vector vk published in the coin setup, and, if it is invalid, sends a $\textsc{complaintKeyShare}$ message to $\mathcal{C}$ (lines 74–75) and returns. If the check passes, it creates a coin share using the threshold-coin scheme CF (line 76) and sends to the committee $\mathcal{C}$. All complaints are verifiable: $\textsc{complaintEncryption}$ is valid if the zk-proof W_j, proving that the published e_j decrypts to (sk_j, r_j)), is valid, and the re-encryption of (sk_j, r_j) gives something different from e_j (lines 81–84). $\textsc{complaintShare}$ is valid if the re-encryption of (sk_j, r_j) gives the published e_j and the key share sk_j is deemed invalid by the CF scheme (lines 85–88). Party P_i outputs in two cases, whichever comes first. First, upon collecting $\tau_{\text{COIN}} + 1$ valid coin shares (line 89), in which case the value of the coin is reconstructed using the underlying CF scheme. Second, upon receiving a valid complaint (line 94), in which case a $\perp$ value is output. The wHDCFVerify check can be implemented by a function that when given a complaint checks if it is valid according to the activation rules in line 81 or line 85, and when given a set of shares checks that they are valid and reruns the activation rule in line 89.

6 Weak Multiple-Dealer Coin-Flip

In this section we define the weak multiple-dealer coin-flip (wMDCF) protocol. It is *weak* as it inherits the agreement property from wHDCF: parties may output $\perp$, but if two honest parties output a value in $\{0, 1\}$, then it will be the same. It is called *multiple-dealer* as there are multiple dealers, forming a *proposers committee*, selected pseudorandomly using $\mathsf{SampleCommittee}_p()$. The protocol uses parameters m_{wMDCF} and w_{wMDCF}. Parameter m_{wMDCF} refers to the size of the proposers committee, i.e., the number of parties that are

Algorithm 6 Scheme wHDCF, algorithm Flip (cid), where an instance sid of wHDCF is created at point p on Ledger. Code for process P_i, P_i is one of the H_j in the coin-setup $(\mathsf{sid}, \mathsf{vk}, (\mathsf{H}_1, e_1), \dots, (\mathsf{H}_{n_{\text{COIN}}}, e_{n_{\text{COIN}}}))$ published on Ledger.

```
    State:
64:     $\mathsf{validShares}[\mathsf{sid}][\mathsf{cid}] \leftarrow [\,]$, for each sid and cid
65:     $\mathsf{justifiedComplaint}[\mathsf{sid}][\mathsf{cid}] \leftarrow \perp$, for each sid and cid
66:     $\mathsf{terminated}[\mathsf{sid}][\mathsf{cid}] \leftarrow 0$, for each sid and cid

67: upon input $(\text{FLIP}, \mathsf{sid}, \mathsf{cid})$ such that $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$ do
68:     $(\mathsf{vk}, (\mathsf{H}_1, e_1), \dots, (\mathsf{H}_{n_{\text{COIN}}}, e_{n_{\text{COIN}}})) \leftarrow \mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger})$
69:     let $\mathcal{C} = \{\mathsf{H}_1, \dots, \mathsf{H}_{n_{\text{COIN}}}\}$
70:     $(\mathsf{sk}'_i, r'_i) = \mathsf{PKE.Dec}_{\mathsf{dk}_i}(e_i)$
71:     if $e_i \neq \mathsf{PKE.Enc}_{\mathsf{ek}_i}((\mathsf{sk}'_i, r'_i))$ then
72:         create zk-proof $\mathsf{W}_i$ that $e_i$ decrypts to $(\mathsf{sk}'_i, r'_i)$
73:         send $(\text{COMPLAINTENCRYPTION}, \mathsf{sid}, \mathsf{cid}, \mathsf{W}_i, \mathsf{sk}'_i, r'_i)$ to parties in $\mathcal{C}$; return
74:     if $\mathsf{CF.VerifyKeyShare}(\mathsf{vk}, i, \mathsf{sk}'_i) = 0$ then
75:         send $(\text{COMPLAINTKEYSHARE}, \mathsf{sid}, \mathsf{cid}, \mathsf{sk}'_i, r'_i)$ to parties in $\mathcal{C}$; return
76:     $\mathsf{s}_i = \mathsf{CF.CreateShare}(\mathsf{sk}'_i, \mathsf{cid})$
77:     send $(\text{COINSHARE}, \mathsf{sid}, \mathsf{cid}, \mathsf{s}_i)$ to parties in $\mathcal{C}$

78: upon deliver $(\text{COINSHARE}, \mathsf{sid}, \mathsf{cid}, s_j)$ from $\mathsf{P}_j$ do
79:     if $\mathsf{CF.VerifyCoinShare}(\mathsf{vk}, \mathsf{cid}, s_j) = 1$ then
80:         append $s_j$ to $\mathsf{validShares}[\mathsf{sid}][\mathsf{cid}]$

81: upon deliver $c = (\text{COMPLAINTENCRYPTION}, \mathsf{sid}, \mathsf{cid}, \mathsf{W}_j, \mathsf{sk}_j, r_j)$ do
82:     $e'_j \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}_j}((\mathsf{sk}_j, r_j))$
83:     if $e'_j \neq e_j$ and $\mathsf{W}_j$ is valid then
84:         $\mathsf{justifiedComplaint}[\mathsf{sid}][\mathsf{cid}] \leftarrow c$

85: upon deliver $c = (\text{COMPLAINTKEYSHARE}, \mathsf{sid}, \mathsf{cid}, \mathsf{sk}_j, r_j)$ do
86:     $e'_j \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}_j}((\mathsf{sk}_j, r_j))$
87:     if $e'_j = e_j$ and $\mathsf{CF.VerifyKeyShare}(\mathsf{vk}, j, \mathsf{sk}_j) = 0$ then
88:         $\mathsf{justifiedComplaint}[\mathsf{sid}][\mathsf{cid}] \leftarrow c$

89: upon $|\mathsf{validShares}[\mathsf{sid}][\mathsf{cid}]| = \tau_{\text{COIN}} + 1$ and $\mathsf{terminated}[\mathsf{sid}][\mathsf{cid}] = 0$ do
90:     let $\left(s_{j_1}, \dots, s_{j_{\tau_{\text{COIN}}+1}}\right) = \mathsf{validShares}[\mathsf{sid}][\mathsf{cid}]$
91:     $s \leftarrow \mathsf{CF.Combine}(\mathsf{cid}, \{s_{j_k}\}_{k \in [\tau_{\text{COIN}}+1]})$
92:     $\mathsf{terminated}[\mathsf{sid}][\mathsf{cid}] \leftarrow 1$
93:     output $(\text{DONE-FLIP}, \mathsf{sid}, \mathsf{cid}, s, \mathsf{validShares}[\mathsf{sid}][\mathsf{cid}])$

94: upon $\mathsf{justifiedComplaint}[\mathsf{sid}][\mathsf{cid}] \neq \perp$ and $\mathsf{terminated}[\mathsf{sid}][\mathsf{cid}] = 0$ do
95:     $\mathsf{terminated}[\mathsf{sid}][\mathsf{cid}] \leftarrow 1$
96:     output $(\text{DONE-FLIP}, \mathsf{sid}, \mathsf{cid}, \perp, \mathsf{justifiedComplaint}[\mathsf{sid}][\mathsf{cid}])$
```

selected to act as a dealer in an instance of wMDCF. Parameter w_{wMDCF} refers to the number of parties in the proposers committee we asynchronously wait for. In Section 7 we show how to set these parameters, such that at least one good setup appears on the ledger, except with negligible probability, and a constant rate γ of the setups are good.

Syntax. The syntax of weak Multiple-Dealer Coin-Flip (wMDCF) is as follows:

Deal On input $(\text{DEAL}, \mathsf{sid})$ a participating party starts running the dealing protocol and may as a result help produce a public output PubOutMultiDeal.

Flip On input $(\textsc{flip}, \mathsf{sid}, \mathsf{cid})$ for a coin identifier cid, after $\mathsf{PubOutMultiDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \bot$, a party starts running the coin-flip protocol and outputs $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, s)$, where $s \in \{\bot\} \cup \{0,1\}^\lambda$.

Security. The security properties of honest-dealer coin-flip are as follows. For the *agreement* and *unpredictability* properties we use a probability $\gamma > 0$, called the *good-setup probability*, which depends on the parameter w_{wMDCF} and on the hiding probability β of wHDCF, and is constant and independent of sid and cid.

Termination

(1) If all honest parties get input $(\textsc{deal}, \mathsf{sid})$ then eventually there is public output $\mathsf{PubOutMultiDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \bot$, except with negligible probability.

(2) If all honest parties get input $(\textsc{flip}, \mathsf{sid}, \mathsf{cid})$ then eventually all honest parties give an output $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, \cdot)$, except with negligible probability.

γ-Agreement For each session sid and coin identifier cid it holds that, if two honest parties output $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{P}})$ and $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{Q}})$, such that $c_{\mathsf{P}} \neq \bot$ and $c_{\mathsf{Q}} \neq \bot$, then $c_{\mathsf{P}} = c_{\mathsf{Q}}$, except with negligible probability. Moreover, with probability γ it holds that no honest party outputs $\bot$ as the value of the coin. All together, this means that, if two honest parties have outputs $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{P}})$ and $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{Q}})$, then $c_{\mathsf{P}} = c_{\mathsf{Q}} \neq \bot$ with probability γ.

γ-Unpredictability For each session sid and coin identifier cid it holds that the value of coin cid is unpredictable with probability γ.

In the full version of this work [3] we formalize the *agreement* and *unpredictability* properties and show the proofs.

Algorithm 7 Scheme wMDCF, algorithm Deal, where an instance sid of wMDCF is created at some point p on Ledger. Code for process P_i.

```
    State:
97:     setups[$w_{\mathsf{wMDCF}}$] ← [ ]

98: upon input $(\textsc{deal}, \mathsf{sid})$ do
99:     $\mathcal{C} \leftarrow \mathsf{SampleCommittee}(\mathsf{sid}, m_{\mathsf{wMDCF}})$
100:    for $j \in [m_{\mathsf{wMDCF}}]$ such that $\mathcal{C}[j] = P_i$ do
101:        $\mathsf{wHDCF}(\mathsf{Deal}, ((\mathsf{P}_i, j), \mathsf{sid}))$

102: upon $w_{\mathsf{wMDCF}}$ setups $\mathsf{PubOutMultiDeal}(((\mathsf{P}_j, k), \mathsf{sid}), \mathsf{Ledger}) \neq \bot$ where $\mathcal{C}[k] = \mathsf{P}_j$
103:    Let setups contain the identifiers which gave public output sorted deterministically
104:    $\mathsf{seed}(\textsc{seed}, \mathsf{sid})$
```

Construction. On Algorithm 7 we implement Deal. On input $(\textsc{deal}, \mathsf{sid})$, a protocol instance is created with some starting point p. For each time P_i is sampled to be a dealer in a wHDCF instance (line 99), it creates a new instance of wHDCF and runs the Deal algorithm. Every party waits for w_{wMDCF} instances of the wHDCF protocol (started by the dealers sampled in line 99) to give public output on the Ledger. When this happens, parties run an instance of the seed protocol (line 104). This seed will be later used in the Flip algorithm of wMDCF to pseudorandomly choose one of the w_{wMDCF} setups. We define $\mathsf{PubOutMultiDeal} = \mathsf{PubOutSeedOpen}$, so the output of the seed protocol signals the end of the dealing phase.

Algorithm 8 Scheme wMDCF, algorithm Flip (cid), where an instance sid of wMDCF is created at some point p on Ledger. Code for process P_i.

```
105: upon input $(\textsc{flip}, \mathsf{sid}, \mathsf{cid})$ such that $\mathsf{PubOutMultiDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$ do
106:     $j \leftarrow H(\mathsf{sid}, \mathsf{cid}, \mathsf{seed})$
107:     $\mathsf{wHDCF}(\textsc{flip}, \mathsf{setups}[j], \mathsf{cid})$

108: upon deliver $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, s, \pi)$
109:     if $\mathsf{wHDCFVerify}(\pi, s)$ then
110:         output $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, s)$
```

In Algorithm 8 we implement Flip. On input $(\textsc{flip}, \mathsf{sid}, \mathsf{cid})$ and after observing public output PubOutMultiDeal every party P_i uses a cryptographic hash function H, to hash $(\mathsf{sid}, \mathsf{cid}, \mathsf{seed})$ into $j \in \{1, \dots, w_{\mathsf{wMDCF}}\}$ (line 106). Then, the algorithm Flip of the wHDCF_j instance is used to compute the value of coin cid. We assume that each party on the committee of the selected wHDCF instance disseminate the output to the ground population.

7 Setting the Parameters

▶ **Definition 2** (Binomial distribution). *Let X a random variable counting the number of successes out of n trials, where success happens with probability p. Then X follows the binomial distribution, i.e., $X \sim \mathcal{B}(n,p)$ and the probability that exactly k successes happen is*

$$\Pr[X = k] = \Pr[\mathcal{B}(n,p) = k] = \binom{n}{k} p^k (1-p)^{(n-k)}. \tag{2}$$

7.1 Sampling a holding committee for wVSS and wHDCF

Let n denote the size of a holding committee and $\tau < n/2$ denote a number, such that the holding committee has at most τ corruptions with a constant probability β, and more than $n - \tau$ corruptions only with a negligible probability $\epsilon = 2^{-\lambda}$, where λ is the security parameter. The idea is the following. If we use a (n, τ)-secet-sharing or common-coin scheme in the committee, then the committee is *hiding* with probability β and *live* with probability $1 - \epsilon$. These capture the parameters of both the wVSS and the wHDCF schemes. In wVSS we have $n \triangleq n_{\mathrm{VSS}}$ and $\tau \triangleq \tau_{\mathrm{VSS}}$, and in wHDCF we have $n \triangleq n_{\mathrm{COIN}}$ and $\tau \triangleq \tau_{\mathrm{COIN}}$.

As discussed earlier, we model a committee-election mechanism as a black-box function SampleCommittee(), which samples parties with probability proportional to their stake at some well-defined point on the ledger. As SampleCommittee() does sampling with replacement, it can be modelled with a binomial distribution. Using (2) we have that $\beta = \sum_{k=0}^{\tau} \Pr[\mathcal{B}(n, 1-p) = k]$ and $\epsilon = \sum_{k=n-\tau+1}^{n} \Pr[\mathcal{B}(n, 1-p) = k]$, for $p = 2/3$. In Table 1 we show various combinations for n and τ, such that $\epsilon \leq 2^{-\lambda}$ for $\lambda = 60$, and the resulting hiding probability β.

7.2 Sampling a proposer committee for seed and wMDCF

In protocols seed and wMDCF parties have a chance to participate in the *proposers committee*, i.e., to win the right to become a dealer in a wVSS or wHDCF instance, respectively. Parties are again sampled using SampleCommittee (Section 7.1), which returns a committee of size

m, but the protocols only wait for the first w setups to appear on Ledger and only use those. In seed, we have $m \triangleq m_{\text{SEED}}$ and $w \triangleq w_{\text{SEED}}$, and in wMDCF we have $m \triangleq m_{\mathsf{wMDCF}}$ and $w = w_{\mathsf{wMDCF}}$.

Necessary conditions. As before, we need to make sure that, except with negligible probability $\epsilon = 2^{-\lambda}$, there are at least w honest parties on the committee to ensure termination. This is bounded as ϵ in Section 7.1 but with n and τ replaced by m and $w - 1$ respectively. But now we additionally need to make sure that, except with negligible probability at least one of the w setups that appear on Ledger is a *good setup*, that is, from an honest party who sampled a committee with less than τ corruptions. This condition corresponds exactly to the setup in Section 7.1 being hiding, but with the probability p changed to account for the fact that we are interested in the probability of an honest party *who provided a good setup*. Since an honest dealer has a β (which depends on the parameters of the subprotocol) probability of providing a good setup, we set $p = \beta \cdot 2/3$ and require $\sum_{k=0}^{w-1} \Pr[\mathcal{B}(m, 1-p) = k] \geq 1 - 2^{-\lambda}$.

Good-setup probability. Finally, specifically for wMDCF, we calculate the probability γ, defined in Section 6, that a setup published on Ledger is good, i.e., the probability of getting an unpredictable and agreed upon value in each coin flip. We derive this from the expected number of bad setups, which (by linearity of expectation) is $m \cdot (1 - \beta \cdot \frac{2}{3})$, and from the fact that the adversary can schedule the order of messages, causing all bad setups and, hence, only $w - m \cdot (1 - \beta \cdot \frac{2}{3})$) good setups, to appear on Ledger. This gives us the fraction of good setups that in expectation appear on the ledger as

$$\gamma = \frac{w - (m \cdot (1 - \beta \cdot \frac{2}{3}))}{w}. \tag{3}$$

Putting it all together. We show the resulting parameters with $\lambda = 60$ bits of security in Table 1. As an example, for a holding committee with size $n = 259$ and reconstruction threshold $\tau = 103$, we get hiding probability $\beta = 98.7\%$. Then we can sample a proposers committee of size $m = 653$ and wait for $w = 327$. This results in $84,693$ encrypted shares being posted on the Ledger, and for wMDCF it gives a good-setup probability $\gamma = 31.8\%$.

8 Analysis of Communication Complexity

To demonstrate the power of being able to sample concretely small committees, we analyze the concrete complexity of our protocols. Note that a purely asymptotic analysis would not show any gains over simply using a state of the art ADKG protocol with subset sampling and near optimal resilience. We give all sizes in *bits*, but for simplicity we treat group and field elements as λ bits. For instance, we use 3λ as the size of an encrypted share, which (using Section 2.2.1) consists of 2 group elements and a symmetrically encrypted share of a secret of size λ. This would not be precise for concrete instantiations, but it would only change our estimates by a small constant factor which depends, for example, on the concrete curves being employed.

We define ATOB complexity as the cost of including a message of a given size in Ledger. In the following "broadcast" refers to broadcasting through the ATOB and "multicast" refers to a party sending a message to all parties. As the communication cost of a broadcast and multicast depend on the implementation, we keep these costs opaque and report results as a number of broadcasts and multicasts required. For inter-committee communication we assume point-to-point channels and give results in total number of bits sent.

Table 1 This table shows possible values (subject to conditions in Section 7.1) for the *holding committee* parameters, n and τ, and the resulting hiding probability β. For each obtainable β, it shows possible values (subject to conditions in Section 7.2) for the *proposers committee* parameters, m and w, and the resulting good-setup probability γ. In both seed and wMDCF schemes, each of the w dealers encrypts keys for a committee of size n, which gives a total of $m * w$ encryptions.

n	τ	β	m	w	γ	$n \cdot w$
653	320	$> 1 - 2^{60}$	653	321	32.2%	$209.6K$
300	125	99.9%	653	322	32.3%	$96.6K$
280	114	99.6%	653	323	32.1%	$90.4K$
275	111	99.4%	653	324	32.0%	$89.1K$
271	109	99.3%	653	325	32.0%	$88.1K$
265	106	99.0%	653	326	31.9%	$86.4K$
261	104	98.8%	653	327	31.9%	$85.3K$
259	103	98.7%	653	327	31.8%	$84.7K$
257	102	98.6%	659	330	31.6%	$84.8K$
256	101	98.3%	672	337	31.3%	$86.3K$
254	100	98.1%	682	342	31.1%	$86.9K$
252	99	98.0%	692	347	30.8%	$87.4K$

The wVSS protocol has an ATOB complexity of 1 message of size $n_{\text{VSS}} \cdot 3\lambda + \lambda$ with the encrypted shares and masked message in the setup. To distribute the output either the secret of size λ is multicast or a complaint of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$ (in case of correctly encrypted shares reconstructing an inconsistent setup) is multicast. A priori, every member of the committee needs to multicast the output, giving a multicast complexity of n_{VSS} messages of size λ (or of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$, in which case the dealer can be slashed). The remaining interaction consists of n_{VSS} committee members sending one message with a decrypted share or a complaint of total size at most 4λ to the rest of the committee, resulting in a total message complexity of at most ${n_{\text{VSS}}}^2 \cdot 4\lambda$.

The seed protocol does not add any interaction besides running m_{SEED} instances of wVSS. Only the first w_{SEED} of those to make it onto the ledger will result in interaction between committe members, so the communication complexity is at most $m_{\text{SEED}} \cdot {n_{\text{VSS}}}^2 \cdot 4\lambda$. The ATOB complexity of the deal phase of seed is m_{SEED} messages of size $n_{\text{VSS}} \cdot 3\lambda + \lambda$. To disseminate the outputs there is an additional $(w_{\text{SEED}} - s) \cdot n_{\text{VSS}}$ multicasts of size λ and $s \cdot n_{\text{VSS}}$ multicasts of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$, where s is the number of parties that can be slashed.

The wHDCF protocol has an ATOB complexity of 1 message of size $n_{\text{COIN}} \cdot 3\lambda + \lambda$ and no additional communication in the initial setup phase. The message complexity of each coin flip is at most ${n_{\text{COIN}}}^2 \cdot 4\lambda$ to reconstruct the coin (or $\perp$) in the committee, and then to disseminate the value to the ground population each committee member multicasts the reconstructed coin or a complaint of size at most 4λ, resulting in at most ${n_{\text{COIN}}}^2 \cdot 4\lambda$ bits communicated in addition to n_{COIN} multicasts of size 4λ. The deal phase of the wMDCF protocol has the same complexity as m_{wMDCF} deal phases of wHDCF and a single run of the seed protocol. That is, an ATOB complexity of m_{wMDCF} messages of size $n_{\text{COIN}} \cdot 3\lambda + \lambda$ and m_{SEED} messages of size $n_{\text{VSS}} \cdot 3\lambda + \lambda$, and a multicast complexity of $w_{\text{SEED}} \cdot n_{\text{VSS}}$ messages of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$, in addition to a communication complexity of $m_{\text{SEED}} \cdot {n_{\text{VSS}}}^2 \cdot 4\lambda$ bits. Whenever a coin needs to be flipped using wMDCF, the message complexity is that of running the selected wHDCF protcol.

To refresh the setup after the stake distribution has changed, one would need to first run an instance of the seed protocol and then the deal phase of the wMDCF protocol. Using the best parameters in Table 1 the concrete cost of refreshing the setup is 1959 messages of size 778λ, and $169,386$ multicasts of size at most 208λ. The communication complexity of flipping a coin and disseminating it to all parties is $259^2 \cdot 4\lambda$ in addition to 259 multicasts of 4λ bits. Employing the optimizations in Remark 3 reduces the multicast complexity of refreshing the setup to 654 messages of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$. Similarly the cost of distributing a coin becomes the same as 1 multicast of size 4λ.

If we were to assume $t < 0.3n$ in the paradigm of "subset sampling with almost optimal resilience" [5], and need a committee with honest supermajority with probability $1-2^{-60}$, then one would need to sample a committee with 16037 parties [22, Table 1]. If we then instantiate a state of the art ADKG protocol with $O(n^3\lambda)$ communication using the committee, assuming for the sake of an example the concrete cost is $n^3\lambda$, then we get a complexity of $> 4 \cdot 10^{12}\lambda$. It is then clear that our approach is far cheaper for all but extremely large values of n.

▶ Remark 3 (Deduplicating multicasts). Notice that each party only needs to receive a single proof of output for each of the w_{SEED} wVSS setups. Since large-scale P2P networks usually employ gossiping with deduplication of previously forwarded messages, each node can consider different output justifications from the same wVSS as identical for the purpose of decuplication. We conjecture that in most gossip-based P2P networks this results in a communication cost which is less than that of a single multicast, as it can be seen as a multicast from a single source which has gotten a headstart by being predistributed to $O(\lambda)$ nodes. With this instantiation the cost of disseminating the wVSS outputs from the seed protocol becomes the same as multicasting w_{SEED} messages of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$ over the gossip network. The same deduplication trick can be employed when disseminating the coin flips, reducing the cost of distributing the coin to the same as 1 multicast of size 4λ.

9 Conclusion

In this work we have presented protocols for generating randomness in an asynchronous PoS setting with dynamic participation. They are practical and concretely efficient, employ no trusted setup, and they make use of small committees. We have computed concrete numbers for the committees. Specifically, we can have a committee of $m = 653$ proposers, each generating a setup for $n = 359$ holders, resulting in approx. $85K$ encrypted values posted on Ledger. For $\kappa = 60$ bits of security and assuming optimal corruption $1/3$ in the ground population, our protocols are live with all but negligible probability. Our common-coin protocol is unpredictable and agreed-upon with probability approx. 31.8%, and, as it is based on threshold cryptography, the setup can be used for a flipping a polynomial number of coins. These committee sizes result from the fact that we require not all but only a constant factor of our setups to be good.

References

1 Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation. *IACR Cryptol. ePrint Arch.*, page 1759, 2022.

2 Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *PODC*, pages 363–373. ACM, 2021.

3 Orestis Alpos, Christian Cachin, Simon Holmgaard Kamp, and Jesper Buus Nielsen. Practical large-scale proof-of-stake asynchronous total-order broadcast. *IACR Cryptol. ePrint Arch.*, page 1103, 2023.

4 Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC*, pages 27–30. ACM, 1983.

5 Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In *TCC (1)*, volume 12550 of *Lecture Notes in Computer Science*, pages 353–380. Springer, 2020.

6 Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

7 Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO (1)*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

8 Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.

9 Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2001.

10 Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptol.*, 18(3):219–246, 2005.

11 Jan Camenisch, Manu Drijvers, Timo Hanke, Yvonne-Anne Pignolet, Victor Shoup, and Dominic Williams. Internet computer consensus. In *PODC*, pages 81–91. ACM, 2022.

12 Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC*, pages 42–51. ACM, 1993.

13 Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In *ACNS*, volume 10355 of *Lecture Notes in Computer Science*, pages 537–556. Springer, 2017.

14 Ignacio Cascudo and Bernardo David. ALBATROSS: publicly attestable batched randomness based on secret sharing. In *ASIACRYPT (3)*, volume 12493 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.

15 Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. ALGORAND AGREEMENT: super fast and partition resilient byzantine agreement. *IACR Cryptol. ePrint Arch.*, page 377, 2018.

16 Kevin Choi, Arasu Arun, Nirvan Tyagi, and Joseph Bonneau. Bicorn: An optimistically efficient distributed randomness beacon. *IACR Cryptol. ePrint Arch.*, page 221, 2023.

17 Kevin Choi, Aathira Manoj, and Joseph Bonneau. Sok: Distributed randomness beacons. *IACR Cryptol. ePrint Arch.*, page 728, 2023.

18 Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement WHP. In *DISC*, volume 179 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

19 Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. Spurt: Scalable distributed randomness beacon with transparent setup. In *IEEE Symposium on Security and Privacy*, pages 2502–2517. IEEE, 2022.

20 Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *IEEE Symposium on Security and Privacy*, pages 2518–2534. IEEE, 2022.

21 Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98. Springer, 2018.

22 Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In *CCS*, pages 683–696. ACM, 2022.

23 Drand. A distributed randomness beacon daemon, 2022. URL: `https://drand.love`.

24 Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–437. IEEE Computer Society, 1987.

25 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In *PODS*, pages 1–7. ACM, 1983.

26 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68. ACM, 2017.

27 Valerie King and Jared Saia. Byzantine agreement in expected polynomial time. *J. ACM*, 63(2):13:1–13:21, 2016.

28 Valerie King and Jared Saia. Correction to byzantine agreement in expected polynomial time, JACM 2016. *CoRR*, abs/1812.10169, 2018.

29 Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. *IACR Cryptol. ePrint Arch.*, page 366, 2015.

30 Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130. IEEE Computer Society, 1999.

31 Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Asynchronous byzantine agreement with optimal resilience. *Distributed Comput.*, 27(2):111–146, 2014.

32 Protocol Labs. Filecoin: A decentralized storage network. `https://filecoin.io/filecoin.pdf`, 2017.

33 Michael O. Rabin. Randomized byzantine generals. In *FOCS*, pages 403–409. IEEE Computer Society, 1983.

34 Mayank Raikwar and Danilo Gligoroski. Sok: Decentralized randomness beacon protocols. In *ACISP*, volume 13494 of *Lecture Notes in Computer Science*, pages 420–446. Springer, 2022.

35 David A. Wagner Ronald L. Rivest, Adi Shamir. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, 1996.

36 Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R. Weippl. Hydrand: Efficient continuous distributed randomness. In *IEEE Symposium on Security and Privacy*, pages 73–89. IEEE, 2020.

37 Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

38 Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE Symposium on Security and Privacy*, pages 444–460. IEEE Computer Society, 2017.