Modeling Resources in Permissionless Longest-Chain Total-Order Broadcast

Sarah Azouvi
Protocol Labs

Christian Cachin
University of Bern, Switzerland

Duc V. Le
University of Bern, Switzerland

Marko Vukolić
Protocol Labs

Luca Zanolini
University of Bern, Switzerland

Abstract

Blockchain protocols implement total-order broadcast in a permissionless setting, where processes can freely join and leave. In such a setting, to safeguard against Sybil attacks, correct processes rely on cryptographic proofs tied to a particular type of *resource* to make them eligible to order transactions. For example, in the case of Proof-of-Work (PoW), this resource is computation, and the proof is a solution to a computationally hard puzzle. Conversely, in Proof-of-Stake (PoS), the resource corresponds to the number of coins that every process in the system owns, and a secure lottery selects a process for participation proportionally to its coin holdings.

Although many resource-based blockchain protocols are formally proven secure in the literature, the existing security proofs fail to demonstrate why particular types of resources cause the blockchain protocols to be vulnerable to distinct classes of attacks. For instance, PoS systems are more vulnerable to long-range attacks, where an adversary corrupts past processes to re-write the history, than PoW and Proof-of-Storage systems. Proof-of-Storage-based and PoS-based protocols are both more susceptible to private double-spending attacks than PoW-based protocols; in this case, an adversary mines its chain in secret without sharing its blocks with the rest of the processes until the end of the attack.

In this paper, we formally characterize the properties of resources through an abstraction called *resource allocator* and give a framework for understanding longest-chain consensus protocols based on different underlying resources. In addition, we use this resource allocator to demonstrate security trade-offs between various resources focusing on well-known attacks (e.g., the long-range attack and nothing-at-stake attacks).

2012 ACM Subject Classification Theory of computation → Cryptographic protocols; Software and its engineering → Distributed systems organizing principles

Keywords and phrases blockchain, consensus, resource, broadcast

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2022.19

Related Version *Full Version*: https://arxiv.org/abs/2211.12050

Funding DVL has been supported by a grant from Protocol Labs to the University of Bern. LZ has been supported by the Swiss National Science Foundation (SNSF) under grant agreement Nr. 200021_188443 (Advanced Consensus Protocols).

Acknowledgements The authors thank anonymous reviewers for helpful feedback.

26th International Conference on Principles of Distributed Systems (OPODIS 2022).
Editors: Eshcar Hillel, Roberto Palmieri, and Etienne Rivière; Article No. 19; pp. 19:1–19:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Permissionless consensus protocols are open for everyone to participate and often rely on a *resource* to protect against Sybil attacks. In the case of Proof-of-Work (PoW), this resource is computation: A computational puzzle must be solved in order to gain writing rights in the system. In contrast, in a Proof-of-Stake (PoS) system, writing access is granted using a form of lottery where participants are elected proportionally to the number of coins they own. Other resource-based systems, such as Proof-of-Storage, have also appeared. Participants are elected proportionally to the number of resources they commit to the system, and hence this commitment must be publicly verifiable. Different resources present different trade-offs. For example, PoS is much more energy-efficient than PoW but presents many additional vulnerabilities [5]. Comparing the security of protocols based on multiple resource types is a non-trivial task, as they use different assumptions and frameworks.

In this paper, we provide a common framework to formally compare consensus protocols based on different underlying resources. We only consider longest-chain protocols [11] that rely on an underlying resource, as we want to highlight the properties affected by varying the resource for the same consensus method. In future work, our framework could be used to model further approaches to ensure consensus, such as well-known BFT protocols [7], for instance. In longest-chain protocols, one participant is elected at each time step, on expectation, proportionally to their amount of resource and that participant gets to write to the append-only database by adding a *block* containing all the necessary data to the longest chain of blocks.

We also explore known attacks in this work. The first one is the long-range attack. In a long-range attack, an adversary corrupts processes that used to participate in the system but that no longer hold any resources. Moreover, we investigate nothing-at-stake attacks, where processes mine on multiple chains at the same time, and private attacks, where an adversary mines on its own chain without contributing to the honest chain. We are interested in quantifying gain or loss of security with different resources. It has already been shown that, when considering longest-chain protocols, PoS is less secure than PoW. We furthermore show that Proof-of-Storage stands in the middle, as storage is not virtual (like stake), but is reusable (unlike the computation of PoW).

We start the paper by providing a formal framework in which protocols based on different resources can meaningfully be compared. We differentiate between virtual and external resources to highlight which properties make longest-chain PoS and Proof-of-Storage less secure than PoW, although they both present trade-offs when it comes to their efficiency.

Contributions. Our contributions can be summarized as follows.

- We formally characterize the properties of resources through an abstraction called *resource allocator* and formally define properties for a *secure* resource allocator.
- We concretely define different resource allocator abstractions, each one for every type of resource used in popular blockchain protocols, namely, computation, stake, and storage.
- We present an algorithm that, when instantiated with different resource allocators, leads to a generalization of existing protocols such as Nakamoto consensus, Ouroboros Praos, and Filecoin's consensus protocol. We also show formally that this generalization implements total-ordered broadcast under a fixed total resource in a permissionless setting.
- We demonstrate how different resources lead to different security trade-offs by leveraging our model to explain long-range attacks against virtual resources and attacks related to the nothing-at-stake nature of reusable resources.

Related Work. Since the emergence of Bitcoin in 2008, the academic community has developed a number of frameworks [16, 22, 18, 11] for studying the safety and liveness properties of its Nakamoto consensus protocol. These studies also established a strong foundation for the development of blockchain protocols based on more eco-friendly types of resources, such as stake and storage. However, despite the fact that all resource-based blockchains have been formally proven to be secure, these results have failed to explain why certain properties of resources make some blockchain protocols more susceptible to particular types of attacks than others. To the best of our knowledge, no prior work has attempted to formally study the properties of underlying resources, and our work aims to fill this gap.

Lewis-Pye and Roughgarden [21] present the concept of a *resource pool* that reflects the resource balance of processes in the system at any time, and they use a *permitter* together with the resource pool to abstract away the leader selection procedure. Using this formalization, they demonstrate two crucial impossibility results for permissionless systems. Two main results of their work are: *(i)* no permissionless, deterministic, and decentralized protocol solves the Byzantine Agreement problem in a synchronous setting, and *(ii)* no permissionless and probabilistic protocols solve the Byzantine Agreement problem in the unsized setting (in which the total number of resources is unknown) with partially synchronous communication. However, their work could not capture several aspects of underlying resources used in blockchain protocols; therefore, their work did not demonstrate long-range attacks against virtual resources such as stake, and the cost of several other attacks on *reusable* resources. Our work takes a similar approach of abstracting away the leader selection process with a resource allocator (c.f., Section 3), and we further formalize the properties of resources through the interactions between the process and this allocator. With this formalization, we prove how permissionless and probabilistic blockchain protocols guarantee properties of a total-order broadcast in a synchronous setting and demonstrate various attacks against *virtual* or *reusable* resources.

Terner [25] also investigates how to abstract resources used in permissionless blockchains. While this work outlines several essential properties of resources and studies how the resource generation rate affects the standard properties (i.e., consistency and liveness) of robust transaction ledger, the study does not characterize the properties of the underlying resources used in permissionless blockchain protocols. Consequently, this model fails to explain why distinct types of resources render some protocols vulnerable to certain attacks (e.g., long-range attacks and private attacks).

2 Model and Definitions

2.1 System Model

Time. We assume that the protocol proceeds in *time steps* and define a time step to be a value in $\mathbb{N}$. Moreover, we consider 0 as starting time step of protocol execution.

Processes. We consider a system consisting of a set of *processes*, $\mathcal{P} = \{p_1, p_2, \ldots\}$. Processes interact with each other through exchanging messages. A protocol for $\mathcal{P}$ consists of a collection of programs with instructions for all processes. Moreover, to capture the permissionless nature of various blockchain protocols, processes can join the system at any time. we denote, $\mathcal{P}_{\leq t}$, the set of all processes that have participated in the protocol before the time step t. Hence, $\mathcal{P}_t \subseteq \mathcal{P}_{t'}$ for all $t \leq t'$. At the beginning of each time step, a process becomes *activated*, and it starts to follow a deterministic protocol. This includes processing any messages that may have arrived from other processes. Once done, it becomes *deactivated*. We assume that the activation period of a process p_i starts at the time step t and ends before time step $t + 1$.

Communication. We assume there is a low-level primitive for sending messages over point-to-point links between each pair of processes that know of each other, as well as a probabilistic broadcast primitive [7]. Point-to-point messages are authenticated and delivered reliably among correct processes. In probabilistic broadcast, correct processes *gossip-deliver* and *gossip-broadcast* messages with an overwhelming probability, no message is delivered more than once, and no message is created or corrupted by the network.

Network Delay. We denote by $\Delta \in \mathbb{N}$ with $\Delta \geq 1$ the maximum network delay [14]. Namely, if a correct process *gossip-broadcasts* a message m at a time step t, then other processes will have *gossip-delivered* or received over the message by the beginning of a time step $t + \Delta$ with an overwhelming probability.

Idealized Digital Signature. A digital signature scheme, Σ, consists of two operations, $\mathit{Sign}(\cdot, \cdot)$ and $\mathit{Verify}(\cdot, \cdot, \cdot)$. The operation $\mathit{Sign}(p_i, \cdot)$ invoked by p_i takes $m \in \{0,1\}^*$ as input and returns a signature $\sigma \in \{0,1\}^*$. Only p_i can invoke $\mathit{Sign}(p_i, \cdot)$. The operation $\mathit{Verify}(p_i, \cdot, \cdot)$ takes as input a signature, σ, and a message m; $\mathit{Verify}(p_i, \cdot, \cdot)$ returns TRUE for any $p_i \in \mathcal{P}$ and $m \in \{0,1\}^*$ if and only if p_i has invoked $\mathit{Sign}(p_i, m)$ and obtained σ before. Any process can invoke $\mathit{Verify}(\cdot, \cdot, \cdot)$.

Random Oracle. All hash functions are modeled as a random oracle, H, that can be queried by any process. H takes as input a bit string $x \in \{0,1\}^*$ and returns a uniformly random string from $\{0,1\}^\lambda$ where λ is the security parameter. Also, upon repeated queries, H always outputs the same answer.

2.2 Modeling Blockchain Data Structures

Blocks. We use tx to denote a *transaction*. We write $\overline{\mathsf{tx}} = [\mathsf{tx}_1, \ldots, \mathsf{tx}_m]$ to denote a list of transactions. A *block* is $B = (h, \overline{\mathsf{tx}}, \pi, \sigma_i)$, where h is a hash value, $\overline{\mathsf{tx}}$ is a list of transactions, π is a resource commitment proof (cf. Section 3) and σ_i is a signature on $(h, \overline{\mathsf{tx}}, \pi)$. In this work, we assume that blocks are signed. In this way, we can abstract away the notion of *coinbase* transactions, i.e., the first transaction in a block, created by a miner, and used to collect the block reward. Finally, we denote with $B_0 = (\perp, \overline{\mathsf{tx}}, \perp, \perp)$ the *genesis block*.

Blockchain. A *blockchain* $\mathcal{C} = [B_0, B_1, \ldots]$ with respect to the genesis block B_0 is a chain of blocks forming a hash chain such that $h_j = H(B_{j-1})$ for $h_j \in B_j$ for $j = 1, 2, \ldots$ with $B_j = (h_j, \bar{\mathsf{tx}}_j, \pi_j, \sigma_j)$. For a blockchain $\mathcal{C}$, we use $\mathcal{C}[-k]$ to denote the last k-th block in $\mathcal{C}$, let $\mathcal{C}[k]$ to denote block B_k (i.e., block at height k), and write $\mathcal{C}[:-k]$ to denote the first $|\mathcal{C}| - k$ blocks. $|\mathcal{C}|$ denotes the length of $\mathcal{C}$. We write $\mathcal{C} \preceq \mathcal{C}'$ when $\mathcal{C}$ is a prefix $\mathcal{C}'$. We use $\mathcal{C}^t$ to denote the blockchain at time step t. For two time steps, t_1 and t_2, $\mathcal{C}^{t_2}/\mathcal{C}^{t_1}$ is a set of blocks that is in $\mathcal{C}^{t_2}$ but not in $\mathcal{C}^{t_1}$.

State. The blockchain *state* st specifies different information of the underlying blockchain protocol, e.g., the stake distribution of each process, the block information, such as timestamps, as well as contract local states. The blockchain state st can be reconstructed by executing transactions included in a blockchain $\mathcal{C}$. Without loss of generality, we define the state to be the blockchain, $st = \mathcal{C}$. Also, we write $st = (\mathcal{C}, B)$ to indicate that a block B is potentially appended to $\mathcal{C}$.

Validity. We introduce the notion of *validity* for transactions and blockchains to capture the fact that only "valid" transactions are delivered. More importantly, for all blockchain protocols, the decision on the validity is determined locally by all processes. Because of this, we define the validity as follows. A transaction x is *valid* with respect to $\mathcal{C}$ if tx satisfies a *validation predicate* $\mathbb{P}(\mathcal{C}, \cdot)$ locally known to all processes (i.e., $\mathbb{P}(\mathcal{C}, [\mathsf{tx}]) = \textsc{true}$). We also use $\mathbb{P}(\mathcal{C}, \overline{\mathsf{tx}}) = \textsc{true}$ to indicate that the sequence of transactions in $\overline{\mathsf{tx}}$ is valid (i.e., does not consume the same output in Bitcoin or the same nonce in Ethereum), and we define $\mathbb{P}(\mathcal{C}, [\ ])$ to be $\textsc{true}$. Depending on the blockchain protocol, a *valid block* B issued by p_i should consist of a valid signature issued by p_i, a valid "proof" π for a so-called resource commitment that we introduce in Section 3 and valid transactions with respect to $\mathcal{C}$ such that $\mathcal{C}[-1] = B$, (i.e., $\mathbb{P}(\mathcal{C}, \overline{\mathsf{tx}}) = \textsc{true}$ for $\overline{\mathsf{tx}} \in B$). Finally, *valid blockchains* are chains that consist of only valid blocks and start from the genesis block B_0.

2.3 Total-order Broadcast

We will show that the blockchain protocols considered here guarantee the following properties of total-order broadcast in a permissionless setting. In particular, total-order broadcast ensures that all processes deliver the same set of transactions in a common global order. In total-order broadcast, every process broadcasts a transaction by invoking *a-broadcasts*(tx). The broadcast primitive outputs a transaction tx through an *a-deliver*(tx) event. In this model, we do not distinguish between a process and a client. A client can be considered as a process that only broadcast transactions and does not participate in mining.

▶ **Definition 2.1** (Total-order Broadcast). *A protocol for total-order broadcast satisfies the following properties.*

Validity *If a correct process, p_i a-broadcasts a valid transaction* tx *according to $\mathbb{P}(\cdot,\cdot)$ (i.e., the validation predicate defined in Section 2), then p_i eventually* a-delivers tx *with an overwhelming probability.*

No duplication *No correct process a-delivers the same transaction* tx *more than once.*

Agreement *If a transaction* tx *is a-delivered by some correct process, then with an overwhelming probability* tx *is eventually* a-delivered *by every correct process.*

Total order *Let* tx_1 *and* tx_2 *be any two transactions, and suppose p_i and p_j are any two correct processes that* a-deliver tx_1 *and* tx_2. *If p_i* a-delivers tx_1 *before* tx_2, *then with an overwhelming probability, p_j* a-delivers tx_1 *before* tx_2.

3 Modeling Resources in Blockchain

In this section, we model resources, formalize their properties through the abstraction of a *resource allocator*, and state our threat assumptions. The definition of a *resource allocator* in this section is only syntactic; security and liveness properties of the resource allocator are defined in Section 4.

▶ **Definition 3.1** (Resource Budget). *A resource budget r is a value in $\mathbb{N}$. At any given time, each process p_i has a resource budget r_i. In particular, there exists a function Alloc $: \mathcal{P} \times \mathbb{N} \to \mathbb{N}$ that takes as input a process p_i and a time step t, outputs the resource budget of a process at time step t. We define R to be the fixed resource budget existing in the system.*

The definition of a fixed resource budget and the resource allocation function can be viewed as the sized setting and the resource pool definition in Lewis-Pye and Roughgarden framework [21].

We note that the specification of the resource budget varies depending on protocols; e.g., for PoW, we define the budget to be a number of hash function evaluations per time step. We now define *resource allocator*, an abstraction that will allow us to reason about different resources.

▶ **Definition 3.2** (Resource Allocator). *A resource allocator, RA, interacts with the processes through input events (RA-*commit*, RA-*validate*) and output events (RA-*assign*, RA-*is-committed*):*

- *RA-*commit*(p_i, st, r): At time step t, every process p_i may request a resource commitment π from the resource allocator by invoking RA-*commit *on inputs a state st and a resource budget $0 \leq r \leq Alloc(p_i, t)$, i.e., p_i does not RA-*commit *more resources than it possesses. At the end of the activation period of p_i, the resource allocator either assigns a resource commitment π and a resource budget r to process p_i through an RA-*assign*(p_i, st, r, π) event or assigns an empty value $\perp$ and* possibly *a resource r to p_i through RA-*assign*$(p_i, st, r, \perp)$.*
- *RA-*validate*(p_i, st, π): Every process p_i may validate a resource commitment π by invoking RA-*validate *on input a state, st, and a resource commitment π. The resource allocator validates the resource commitment π, through an event RA-*is-committed*(p_i, st, b) event, with $b =$ TRUE if the commitment π is a valid resource commitment for the state st or $b =$ FALSE otherwise.*

A process triggers *RA-commit* to pledge its resources to a system, and it can be assigned a resource commitment as a result to extend the blockchain. If the resource commitment π is included on-chain, then it must be *valid* (i.e., *RA-validate* returns TRUE) for the block to be accepted. Moreover, we assume that all the events to and from the resource allocator happen within the same time step. In particular, if a process p_i *RA-commit*s some resource budget r at time step t, at the end of the activation period for p_i process p_i will receive either a resource commitment π and r or an empty resource commitment value $\perp$ and r.

Resources can be classified into various types. In our model, these types can be described as the interactions between processes and the resource allocator. The following definition classifies different types of resources used in existing blockchain protocols.

▶ **Definition 3.3** (Types of resource). *A resource can be classified as follows.*

Virtual *A resource is* virtual *when the resource allocator determines the resource budget of all processes from the given blockchain state st. For a virtual resource, we assume that there exists a function StateAlloc : $\mathcal{P} \times \mathbb{C} \rightarrow \mathbb{N}$ that takes as input a process p_i and a blockchain $\mathcal{C}$ and outputs the resource budget of p_i, and p_i can invoke RA-*commit*$(\cdot, \cdot, r)$ on an empty resource, $r = \perp$.*

External *A resource is* external *when a process must allocate the resource* externally *with a budget $r \geq 0$ to invoke RA-*commit*(). For an external resource, this commitment step is equivalent to giving RA access to the external resource with the budget r. Moreover, we assume that processes cannot lie about the resource budget r and commit more than r.*

Burnable *A resource is* burnable *when a process p can trigger* multiple *RA-*commit*$(\cdot, \cdot, r)$ at a time step t, and it retrieves r through RA-*assign*$(\cdot, \cdot, r, \cdot)$ at the end of the activation period for p_i. For all committing events RA-*commit*$(p_i, \cdot, r_i)$ from the same process p_i that occur within a time step t , we require $\sum_{r_i > 0} r_i \leq Alloc(p_i, t)$.*

Reusable *A resource is* reusable *when a process p_i can use the same resource budget $r \leq Alloc(p_i, t)$ to trigger* infinitely many *RA-*commit*$(\cdot, \cdot, r)$ at each time step t, and p_i does not need to retrieve r from the output event RA-*assign*. Hence, for reusable resources, we denote the value of r in the output event RA-*assign*$(\cdot, \cdot, r, \cdot)$ to be $\perp$.*

▶ Remark 3.4. The assumption on external resources is natural because an *external* resource is inherently unforgeable; for instance, in PoW, processes cannot fake this budget as it is the physical limit of the mining hardware. For resources like storage, the resource is the physical hard drive, and r can be thought of as the capacity of the hard drive.

Failures. A process that follows its protocol during an execution is called *correct*. On the other hand, a *faulty* process may crash or deviate arbitrarily from its specification, such processes are also called *Byzantine*. We consider only Byzantine faults in this work. All Byzantine processes are controlled by a probabilistic polynomial-time adversary, $\mathcal{A}$; we write $p_i \in \mathcal{A}$ to denote that a Byzantine process is controlled by $\mathcal{A}$. In this model, we require the adversary to go through the same process of committing resources and getting assigned resource commitments from the allocator. Since the allocator assigns the commitment at the end of the time step, we require a minimum delay between Byzantine processes to be one. We also note that this requirement is only for definitional reasons and can be relaxed by assuming the network delay to be zero for Byzantine processes. However, the concrete parameters on the probability of getting assigned resource commitments for Byzantine processes will need to be adjusted to reflect this assumption, and we leave this to future work.

▶ **Definition 3.5** (Adversarial Resource Budget). *$R_{\mathcal{A}}$ is the maximum adversarial resource budget. For any time step t, it holds that: $\sum_{p_i \in \mathcal{A}} Alloc(p_i, t) \leq R_{\mathcal{A}}$.*

▶ **Definition 3.6** (Corruption). *At any time step t, an adversary $\mathcal{A}$ can allocate a resource budget of $Alloc(p_i, t)$ from $R_{\mathcal{A}}$ to corrupt a process $p_i \in \mathcal{P}_{\leq t}$.*

4 Resource-based Total-order Broadcast

In this section, we define an algorithm for the *resource-based longest-chain total-order broadcast* using a (*probabilistic*) *resource allocator* RA_{res}. We define various properties needed for a secure resource allocator so that the generic algorithm correctly guarantees properties of total-order broadcast. Then, we concretely define three different resource allocators based on three types of resource: computation, stake, and storage to inherently capture three popular (*probabilistic*) blockchain protocols, namely, Nakamoto consensus, Ouroboros Praos, and Filecoin's consensus protocol. However, due to space constraints, the description of the Proof-of-Storage allocator can be found in the full version of the paper.

unordered: set of transaction tx that has been received for execution and ordering
delivered : set of transaction tx that has been executed and ordered
k: common prefix parameter
$\mathcal{B}$: set of received blocks, $B = (h, \overline{\mathsf{tx}}, \pi, \sigma)$, initially containing $B_0 = (\perp, \overline{\mathsf{tx}}, \perp, \perp)$
$\mathbb{C}$: set of valid blockchains derived from $\mathcal{B}$, initially contains one chain $\mathcal{C} = [B_0]$
$\mathcal{C}_{\mathsf{local}}$: local selected blockchain
B_{com}: a RA-*commit*ted block for p_i
At time step t, $r_i = Alloc(p_i, t)$ if r_i is *external*, $r_i \leftarrow \perp$ if r_i is *virtual*

Figure 1 Initial state of a correct process.

4.1 Generic Resource-based Longest-chain Total-order Broadcast

A protocol for resource-based longest-chain total-order broadcast using RA_{res} allows any process p_i to *broadcast* transactions by invoking *a-broadcast*(tx) and to *deliver* those that are valid (according to a validation predicate $\mathbb{P}(\cdot, \cdot)$ and the local chain, $\mathcal{C}_{\mathsf{local}}$) through an *a-deliver*(tx) event. Delivered transactions are *totally ordered* and stored in a list, *delivered*, by every process.

In particular, when a process p_i *a-broadcast*s a transaction, this is gossiped to every process, and eventually every correct process *gossip-delivers* it and stores it in a set *unordered*. Every stored transaction is then considered by p_i.

At any given time, a process may receive new blocks from other processes. Any process p_i can validate the block by invoking *RA-validate* and *RA-assign*ed resource commitment to a process p_j by RA_{res}. Once the resource commitment is validated, the process verifies other components of the block such as signature and transactions and store new blocks in $\mathcal{B}$. Also, if a block B received by other processes does not have a parent (L22), the process can trigger a request message to pull the blockchain $\mathcal{C}$ with B as the tip from other processes. Upon receiving this request message, other processes re-broadcast every blocks in $\mathcal{C}$ with B as the tip (L13-L16). This step is an oversimplified and inefficient version of how blockchain nodes synchronize the chain with others. The goal is to demonstrate that it is feasible to obtain old blocks from other processes.

At any given time, a correct process adopts the longest chain to its knowledge as its local chain $\mathcal{C}_{\mathsf{local}}$, and *extends* with a block B_{com} it wishes to order at the last block of its local chain $\mathcal{C}_{\mathsf{local}}$. Observe that the *Extend* function in Algorithm 1 captures the operation of creating new blocks, usually called *mining*, in blockchain protocols and we refer to the processes in charge of creating blocks as *miners* or *validators*, interchangeably.

In our model, we abstract this *extending* operation as the interaction between the processes and the resource allocator RA_{res}. Namely, to start extending, process p_i needs to allocate a resource r along with the proposed state $(\mathcal{C}_{\mathsf{local}}, B_{\mathrm{com}})$ to the resource allocator RA_{res} through *RA-commit*(). Once RA_{res} *assigns* a resource commitment π, p_i attaches π to the block and gossips the block to other processes. The details of this interaction differ depending on the type of resource and are left for the next subsections. Figure 1 specifies all data structures maintained by a process, and the code for a process is presented in Algorithm 1.

For Algorithm 1 to satisfy the properties of total-order broadcast, the generic resource allocator needs to satisfy various properties, and we define those properties as follows.

▶ **Definition 4.1** (Secure Resource Allocator). *A resource allocator R is secure if it satisfies the following properties:*

Liveness *At a time step t, if a process p_i invokes RA-commit(p_i, st, r) with a state st and a resource budget r then R issues either RA-assign(p_i, st, r, π) or RA-assign$(p_i, st, r, \perp)$ during time step t.*

Validity *If resource commitment π is a valid resource commitment $(i.e., \pi \neq \perp)$ contained in an output event RA-assign(p_i, st, r, π), then any process p_j can invoke RA-validate(p_j, st, π). The resource allocator R outputs RA-is-committed(p_j, st, b) with $b =$ TRUE.*

Use-Once *At any time step t, for any states st, st_1, st_2, any resource budget r, $r_1, r_2 \in \mathbb{N}$ such that $r_1 + r_2 = r$, the probability that RA responds with RA-assign(p_i, st, r, π) with $\pi \neq \perp$ after RA-commit(p_i, st, r) is* greater or equal *to the probability that RA responds* at least one *RA-assign(p_i, st_i, r_i, π) for $i \in \{1, 2\}$ with $\pi \neq \perp$ after* two *RA-commit(p_i, st_1, r_1) and RA-commit(p_i, st_2, r_2).*

Algorithm 1 Resource-based longest-chain total-order broadcast (process p_i).

```
uses
    Resource allocator: $RA_{\mathsf{res}}$
    Probabilistic reliable broadcast: $gossip$
    Validation predicate: $\mathbb{P}(\cdot, \cdot)$
    Random oracle: $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$
    Signature scheme: $\Sigma = (Sign, Verify)$
init
    $Extend(\mathcal{C} = [B_0])$
upon $a\text{-}broadcast(\mathsf{tx})$ do
    invoke $gossip\text{-}broadcast([\textsc{op}, \mathsf{tx}])$
upon $gossip\text{-}deliver$ $([\textsc{op}, \mathsf{tx}])$ do
    $unordered \leftarrow unordered \cup \{\mathsf{tx}\}$
upon $gossip\text{-}deliver$ $([\textsc{request}, B])$ do          ▷ Receive a request for parents of $B$
    if $\exists\, \mathcal{C} \in \mathbb{C}$ s.t. $\mathcal{C}[-1] = B$ then
        forall $B' \in \mathcal{C}$ do                              ▷ Re-send all parents of $B$
            invoke $gossip\text{-}broadcast$ $([\textsc{blk}, B'])$
upon $gossip\text{-}deliver$ $([\textsc{blk}, B])$ s.t. $B = (h, \overline{\mathsf{tx}}, \pi, \sigma_i)$ do
    if $Verify(p_j, h||\overline{\mathsf{tx}}||\pi||sl_j, \sigma_j) \wedge \exists\, \mathcal{C} \in \mathbb{C}$ s.t. $H(\mathcal{C}[-1]) = h \wedge \mathbb{P}(\mathcal{C}, \overline{\mathsf{tx}})$ then
        $st \leftarrow (\mathcal{C}, B)$
        invoke $RA\text{-}validate(p_i, st, \pi)$
    else
        invoke $gossip\text{-}broadcast([\textsc{request}, B])$          ▷ Request for parents of $B$
upon $RA\text{-}is\text{-}committed(p_i, st, \textsc{true})$ s.t. $st = (\mathcal{C}, B)$ do
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{B\}$
    if $|\mathcal{C}| > |\mathcal{C}_{\mathsf{local}}|$ then
        $\mathcal{C}_{\mathsf{local}} \leftarrow \mathcal{C}$                              ▷ Update the local blockchain
        $Extend(\mathcal{C}_{\mathsf{local}})$
upon $RA\text{-}assign(p_i, st, r, \pi)$ s.t. $st = (\mathcal{C}, B = (h, \overline{\mathsf{tx}}, \pi, \perp)), \pi \neq \perp$ do
    $\sigma_i \leftarrow Sign(p_i, h||\overline{\mathsf{tx}}||\pi)$
    $B \leftarrow (h, \overline{\mathsf{tx}}, \pi, \sigma_i)$
    if $r_i$ is $burnable$ then
        $r_i \leftarrow r_i + r$
    invoke $gossip\text{-}broadcast$ $([\textsc{blk}, B])$
upon $RA\text{-}assign(p_i, st, r, \pi)$ s.t. $\pi = \perp$ do
    if $r$ is $burnable$ then
        $r_i \leftarrow r$
    $Extend(\mathcal{C}_{\mathsf{local}})$
upon $a\text{-}deliver$ $([\textsc{op}, \mathsf{tx}])$ do
    $delivered \leftarrow delivered \cup \{\mathsf{tx}\}$
function $Extend(\mathcal{C}_{\mathsf{local}})$
    forall $\mathsf{tx} \in \mathcal{C}_{\mathsf{local}}[: -k] \wedge \mathsf{tx} \notin delivered$ do
        output $a\text{-}deliver([\textsc{op}, \mathsf{tx}])$          ▷ Deliver all transactions in the common prefix
        $unordered \leftarrow unordered \setminus \{\mathsf{tx}\}$
    $h \leftarrow H(\mathcal{C}_{\mathsf{local}}[-1])$
    select a list of transactions $\overline{\mathsf{tx}}$ from $unordered$ such that $\mathbb{P}(\mathcal{C}, \overline{\mathsf{tx}}) = \textsc{true}$
    $B_{\mathsf{com}} \leftarrow (h, \overline{\mathsf{tx}}, \perp, \perp)$
    invoke $RA\text{-}commit(p_i, (\mathcal{C}_{\mathsf{local}}, B_{\mathsf{com}}), r_i)$
    if $r$ is $burnable$ then
        $r_i \leftarrow 0$
```

For a reusable resource*, at any time step t, a resource budget r, a state st and upon potentially multiple repeated* RA-commit(p_i, st, r) *from the same process p_i, if* RA *responds with* RA-assigns(p_i, st, r, π)*, then π is the same for every RA-commit events output by RA.*

Unforgeability *No adversary can produce a resource commitment π such that π has not been previously RA-assigned by RA and, upon* RA-validate*(p_i, st, π), RA triggers RA-is-committed($p_i, st,$ TRUE), for some state st and some process p_i.*

Honest-Majority Assignment *At each time step, we denote with ϱ_H and $\varrho_{\mathcal{A}}$ the probabilities that at least one correct process and one Byzantine process, respectively, obtain a valid resource commitment for each RA-commit. More formally, for every time step t, we define:*

$$\varrho_{\mathcal{A}} = \Pr[\exists RA\text{-}assign(p_i, st, r, \pi) \text{ such that } \pi \neq \bot \wedge p_i \in \mathcal{A}],$$
$$\varrho_H = \Pr[\exists RA\text{-}assign(p_i, st, r, \pi) \text{ such that } \pi \neq \bot \wedge p_i \notin \mathcal{A}].$$

Then we require that:

$$\varrho_{\mathcal{A}} < \frac{1}{\Delta - 1 + 1/\varrho_H}. \tag{1}$$

The *liveness* property aims to capture the mining process in permissionless PoW blockchains and ensure that if processes keep committing resources, *eventually* one process will get assigned the resource commitment to extend the blockchain.

The *validity* property guarantees that a resource commitment can always be verified by any process p_i by triggering at any point *RA-validate*. This property captures the fact that any participant can efficiently verify, for example, the validity of the solution to the computational puzzle in PoW protocols or the evaluation of the verifiable random function in PoS protocols.

The *use-once* property prevents processes from increasing the probability of getting assigned the resource commitment either by committing several times, splitting the resource budget and then committing all the split amounts at different states or by committing a smaller resource budget. Intuitively, the *use-once* property also implies that the property holds for any integer partition of r (i.e., $r = \sum_{r_i>0} r_i$). Moreover, the use-once property also implies that our model mainly focuses on probabilistic protocols as we do not aim to bypass the lower bound established in [21], namely, there is no deterministic protocol in *permissionless* setting that solves consensus. On the other hand, we believe that applying our model to *permissioned* blockchains with PoS, e.g., Tendermint [6], can be interesting future work.

The *unforgeability* property ensures that no process p_i can produce a valid resource commitment π that has not been previously *RA-assigned* by the resource allocator.

Finally, the *honest-majority assignment* implies that despite the network delay, correct processes will have a higher probability of getting assigned the resource commitment at each time step. Equation (1) was established by Gaži *et al.* [18], and it takes into account that honest blocks may get discarded due to the network delay Δ.

Security Analysis. With the defined properties of a secure allocator, our model is equivalent to the idealized model introduced by Gaži *et al.* [18]. Therefore, their result also holds for our protocol, and we present them in our model as follows.

▶ **Lemma 4.2** ([18]). *Algorithm 1 implemented with a secure resource allocator RA_{res} satisfies the following properties:*

Safety *For any time steps t_1 and t_2 with $t_1 \leq t_2$, a common prefix parameter k and any local chain maintained by a correct process $\mathcal{C}_{\mathsf{local}}$, it holds that $\mathcal{C}_{\mathsf{local}}^{t_1}[: -k] \preceq \mathcal{C}_{\mathsf{local}}^{t_2}$ with an overwhelming probability.*

Liveness *For a parameter u and any time step t, let $\mathcal{C}_{\mathsf{local}}$ be the local chain maintained by a correct process, then there is at least one new honest block in $\mathcal{C}^{t+u}/\mathcal{C}^{t}$ with an overwhelming probability.*

Intuitively, *safety* implies that correct processes do not deliver different blocks at the same height, while *liveness* implies that every transaction is eventually delivered by all correct processes. Using Lemma 4.2 and properties of a secure resource allocator, we conclude the following.

▶ **Theorem 4.3.** *If RA_{res} is a* secure *resource allocator, then Algorithm 1 implements total-order broadcast.*

Proof. Observe that, since RA_{res} is a secure resource allocator, it satisfies *use-once* property. Therefore, Byzantine processes cannot amplify the probability $\varrho_{\mathcal{A}}$ by repeatedly triggering *RA-commit()* on reusable resources at the same time step.

For the *validity* property, if a correct process p_i *a-broadcasts* a transaction tx (L9), tx is *gossip-broadcast* (L10) and, after Δ, every correct process *gossip-delivers* tx (L11) and adds it to *unordered* (L12). Eventually, transaction tx is selected by a correct process p_j as part of a block B (L45). Block B is then *gossip-broadcast* by p_j (L33) and eventually, after Δ, every correct process *gossip-delivers* B (L17), validates x (L18), and validates the resource commitment (L20). Observe that, because of the *unforgeability* property of RA_{res}, a valid resource commitment cannot be produced except by the resource allocator. Observe that this last step is possible through the validity property of RA_{res}. The proof then follows from Lemma 4.2.

No duplication property follows from the algorithm; if a correct process p_i *a-delivers* a transaction tx, p_i adds tx to *delivered* and condition at line L41 cannot be satisfied again.

For the *agreement* property, let us assume that a correct process p_i *a-delivered* a transaction tx buried at least k blocks deep in its adopted chain $\mathcal{C}$. Process p_i *a-delivers* a transaction tx when it *updates* the local blockchain with the longest chain $\mathcal{C}$ (L26), tx has not been *a-delivered* yet and tx is part of the common prefix $\mathcal{C}[: -k]$ (L41). The property then follows from Lemma 4.2; eventually every correct process *a-delivers* transaction x, with an overwhelming probability.

Finally, for the *total order* property, from the safety property of Lemma 4.2, we know that correct processes do not deliver different blocks at the same height. This means that at a given height, if two correct processes p_i and p_j *a-delivered* a block, then this block is the same for p_i and p_j with an overwhelming probability. Moreover, since a block is identified by its hash, due to the collision-resistance property of $H(\cdot)$, it also implies that the set and order of transactions included in the block are the same for every correct process. So, if process p_i *a-delivers* transaction tx_1 before tx_2, then either tx_1 and tx_2 are in the same block B with tx_1 appearing before tx_2 or they are in different blocks B_1 and B_2 such that B_2 appears in the chain after B_1. The total-order property follows. ◀

4.2 Proof-of-Work Resource Allocator

In this part, we present the PoW allocator as a concrete instantiation of the resource allocator for *burnable* and *external* resources.

Proof-of-Work Resource Allocator. The PoW resource allocator RA_{pow} is parameterized by ϱ which is the default probability of getting assigned resource commitment for $r = 1$. RA_{pow} works as follows. Upon *RA-commit*(p_i, st, r) by process p_i with a valid chain $\mathcal{C}$ with respect to B_0, RA_{pow} starts r concurrent threads of *Pow*() function which acts as a biased coin with probability ϱ of assigning the resource commitment. Observe that, because computation is a *burnable* and *external* resource, processes cannot lie on about the *committed* resource budget r. In particular, *Pow* uniformly sample a value *nonce* in $\{0,1\}^\lambda$ and either returns $nonce \in \{0,1\}^\lambda$ or $\bot$. If *nonce* is the returned value in $\{0,1\}^\lambda$, then RA_{pow} *assigns* it as the resource commitment to p_i, otherwise it *RA-assigns* $\bot$ to p_i. If the committed chain $\mathcal{C}$ is not valid, then RA_{pow} *RA-assigns* $\bot$ to p_i. Validation of the resource commitment can be done by any process p_j through *RA-validate*; the resource allocator RA_{pow} returns either TRUE or FALSE, depending on the validity of the resource commitment. We implement the resource allocator RA_{pow} in Algorithm 2 and obtain the following lemma and theorem.

Algorithm 2 Implementing PoW resource allocator, RA_{pow}.

```
50: state
51:     $B_0$: Genesis block
52:     $\varrho$: Default probability of getting assigned resource commitment on *one* resource
53: uses
54:     Random oracle: $H : \{0,1\}^* \to \{0,1\}^\lambda$
55: upon $\mathit{RA\text{-}commit}(p_i, st, r)$ s.t. $st = (\mathcal{C}, B), B = (h, \overline{\mathsf{tx}}, \bot, \bot)$ do
56:     if $\mathcal{C}$ is valid $\wedge$ $H(\mathcal{C}[-1]) = h$ then
57:         start $r$ concurrent threads with $nonce_j = \mathit{Pow}(p_i, st)$ for $j \in \{0, \ldots, r-1\}$
58:         wait for all $r$ threads with $\mathit{Pow}(p_i, st)$ for $j \in \{0, \ldots, r-1\}$ to finish
59:         if $\exists$ $nonce_j \neq \bot$ then
60:             output $\mathit{RA\text{-}assign}(p_i, st, r, nonce_j)$
61:         else
62:             output $\mathit{RA\text{-}assign}(p_i, st, r, \bot)$
63:     else
64:         output $\mathit{RA\text{-}assign}(p_i, st, r, \bot)$
65: upon $\mathit{RA\text{-}validate}(p_i, st, \pi)$ s.t. $st = (\mathcal{C}, B), B = (h, \overline{\mathsf{tx}}, \pi, \sigma), \pi = nonce$ do
66:     $b \leftarrow H(h||\overline{x}||nonce) \overset{?}{\leq} \varrho \cdot 2^\lambda$ $\wedge$ $\mathcal{C}$ is valid $\wedge$ $H(\mathcal{C}[-1]) = h$
67:     output $\mathit{RA\text{-}is\text{-}committed}(p_i, st, b)$
68: function $\mathit{Pow}(p_i, st)$                              ▷ With $st = (\mathcal{C}, B)$ and $B = (h, \overline{x}, \bot)$
69:     $nonce \xleftarrow{R} \{0,1\}^\lambda$
70:     if $H(h||\overline{x}||nonce) \leq \varrho \cdot 2^\lambda$ then
71:         return $nonce$
72:     return $\bot$
```

▶ **Lemma 4.4.** *Given the random oracle $H(\cdot)$, the default probability ϱ of getting assigned resource commitment on $r = 1$, and the network delay Δ, there exists a value $R_{\mathcal{A}}$ such that the resource allocator RA_{pow} implemented in Algorithm 2 is a secure resource allocator.*

Proof. *Liveness* property follows from Algorithm 2: upon *RA-commit*(p_i, st, r) from process p_i, the resource allocator RA_{pow} either *(i)* has L56 satisfied and, eventually, outputs *RA-assign*(p_i, st, r, π) with a resource commitment π to p_i or outputs *RA-assign*$(p_i, st, r, \bot)$ to p_i (L61) or *(ii)* if the chain is invalid (L63), and then the allocator outputs *RA-assign*$(p_i, st, r, \bot)$ to p_i.

For the *validity* property, observe that $\pi = nonce$ is valid if and only if there is a valid chain $\mathcal{C}$ with respect to the genesis block B_0 such that the last block $B = \mathcal{C}[-1]$ contains π. Hence, π can be validated by any process p_j through $\mathit{RA\text{-}validate}(p_j, (\mathcal{C}, B^*), \pi)$; RA_{pow} then checks if $H(B^*) = h$ and $H(h||\overline{x}||nonce) \leq \varrho \cdot 2^\lambda$ outputting the same result at any process p_j.

The *use-once* property immediately follows because of the burnable property of the underlying resource (i.e., computation). Since RA_{pow} triggers *RA-assigns* at the end of the the activation period for p_i, we claim that for multiple $\mathit{RA\text{-}commit}(p_i, \cdot, r_i)$ committed by p_i at t, it is equivalent to trigger $\mathit{RA\text{-}commit}(p_i, \cdot, r)$ once for $r = \mathit{Alloc}(p_i, t)$. In particular, at the time step t, let Bad be the event of not getting any resource commitment on all $\mathit{RA\text{-}commit}(p_i, \cdot, r_i)$ for $r_i \in \{r_1, r_2\}$ such that $r = r_1 + r_2$, then the probability of getting the resource commitment is $1 - \Pr[\mathsf{Bad}] = 1 - (1 - 1 + (1 - \varrho)^{r_1})(1 - 1 + (1 - \varrho)^{r_2}) = 1 - (1 - \varrho)^r$.

Since the resource allocator RA_{pow} uses the random oracle H (i.e., idealized hash function with no exploitable weaknesses), the *unforgeability* property follows from the observation that, in order to produce a valid resource commitment, p_i has no better way to find the solution than trying many different queries to H. This implies that p_i has the same probability of obtaining a valid local resource commitment as it would have by *RA-committing* to the resource allocator.

For the *honest-majority assignment* we recall that ϱ_H and $\varrho_{\mathcal{A}}$ are the probabilities that at least one correct process and one Byzantine process get the resource commitment after one *RA-commit*, respectively. In our model, it is not difficult to see that $\varrho_H = 1 - (1 - \varrho)^{R - R_{\mathcal{A}}}$ and $\varrho_{\mathcal{A}} = 1 - (1 - \varrho)^{R_{\mathcal{A}}}$. Therefore, one can easily derive the amount of resource $R_{\mathcal{A}}$ such that $\varrho_{\mathcal{A}} < \frac{1}{\Delta - 1 + 1/\varrho_H}$. ◀

▶ **Theorem 4.5.** *Algorithm 1 with the secure resource allocator RA_{pow} implements total-order broadcast.*

Proof. From Theorem 4.3 and Lemma 4.4, it follows that, since RA_{pow} is a secure resource allocator, then Algorithm 1 with RA_{pow} implements total-order broadcast. ◀

4.3 Proof-of-Stake Resource Allocator

The resource in PoS protocols is the stake of each process, and stake is a *virtual* and *reusable* resource. In those protocols, the probability of a process p_i being assigned a resource commitment is proportional to its stake in the system. In this part, we focus on Ouroboros Praos [9] for our formalization. Before presenting the PoS resource allocator as a concrete instance of a resource allocator for *reusable* and *virtual* resources, we need to introduce additional considerations and definitions.

Reusable Resources. By definition, a reusable resource allows processes to repeatedly trigger *RA-commit* in the same time step using the same resource. Hence, if *RA* does not satisfy *use-once* property and assigns a resource commitment randomly, then every time a process p_i triggers *RA-commit*, process p_i might end up with a different result. For this reason, a naïve implementation of the resource allocator for reusable resources would allow an adversary to amplify the probability of getting assigned a resource commitment (i.e., $\varrho_{\mathcal{A}}$) by repeatedly invoking *RA-commit* using the same resource, i.e., in a grinding attack [3] at the same time step. Hence, for *probabilistic* blockchain protocols, to cope with this problem, one needs to ensure that the allocator satisfies *use-once* property. In particular, from designs of PoS protocols like Snow White [8] and Ouroboros Praos [9], three commonly used approaches to enforce *use-once* property are:

($R1$) **Explicit Time Slots** The first mechanism to enforcing *use-once* property is to index the resource by time slots. Protocols like Ouroboros Praos [9] and Snow White [8] require processes to have synchronized clocks to explicitly track time slots and epochs to ensure that each process derives a deterministic leader selection result from the same state.

($R2$) **Leader Selection from the Common Prefix** This mechanism requires correct processes to extract the set of potential leaders from the common prefix. In particular, the common prefix is a shortened local longest chain that is with overwhelming probability the same for all correct processes. This approach allows them to share the same view of potential leaders.

($R3$) **Deterministic and Trustworthy Source of Randomness** The source of randomness has to be trustworthy to ensure a fair leader election and to defend against an adaptive adversary that might corrupt processes predicted to be leaders for the upcoming time slots. In addition, the source of randomness has to be deterministic for each time slot and chain state in order to prevent the previously mentioned grinding attack. Hence, popular Proof-of-Stake blockchain protocols often rely on sophisticated protocols to produce randomness securely.

Slot and Epoch. An *epoch* e is a set of q adjacent time slot $S = \{sl_0, \dots, sl_{q-1}\}$. In practice, slot sl consists of a sufficient number of time steps so that discrepancies between processes' clocks are insignificant, and processes advance the slot at the same speed. In our model, we simplify this bookkeeping by requiring the allocator to maintain the slot and epoch.

Stake Distribution. The *stake distribution* at a time step t is $\mathbb{S}^t_{\mathsf{stake}} = \{(p_1, r_1), \dots\}$ with $r_i \geq 0$, specifies the amount of stake owned by each process $p_i \in \mathcal{P}_{\leq t}$. We denote $\mathbb{S}^e_{\mathsf{stake}}$ the stake distribution at the beginning of epoch e. The stake distribution $\mathbb{S}^e_{\mathsf{stake}}$ can be obtained from $StateAlloc(\mathcal{C}[0:sl], p_i)$ for $sl \leq e \cdot q$ for each $p_i \in \mathcal{P}_{\leq t}$.

▶ **Definition 4.6** (Leader Selection Process). *A* leader selection process $(\mathcal{D}, F)$ *with respect to a stake distribution* $\mathbb{S}_{\mathsf{stake}} = \{(p_1, r_1), \dots\}$ *is a pair consisting of a distribution* $\mathcal{D}$ *and a deterministic function* F*. When* $\rho \overset{R}{\leftarrow} \mathcal{D}$*, for all* $sl \in \mathbb{N}$*,* $F(\mathbb{S}_{\mathsf{stake}}, sl; \rho)$ *outputs process* p_i *with probability* $1 - (1 - \varrho)^{r_i}$ *where* ϱ *is the probability of assigning resource commitment for* $r = 1$ *for a given slot.*

Proof-of-Stake Resource Allocator. The Proof-of-Stake resource allocator RA_{pos} with the leader selection process $(\mathcal{D}, F)$ works as follows. First, we require that RA_{pos} keeps track of the current epoch and time slot to correctly assign the resource commitment to process p_i for the current slot. RA_{pos} keeps track of the slot through *Timeout* triggered by the *starttimer*() event. This approach is to enforce the first requirement of explicit time slots (R1). Secondly, upon $RA\text{-}commit(p_i, st, r)$ by process p_i with a valid state st in slot sl, RA_{pos} first checks if a random value $\rho \in \mathcal{D}$ for (p_i, st, sl) has been previously sampled; if so, then RA_{pos} picks it; otherwise, a fresh random value is sampled. This requirement ensures a deterministic and trustworthy source of randomness (R3). Then, RA_{pos} obtains the stake distribution of two epochs before, $\mathbb{S}^{e-2}_{\text{stake}}$ from st. This ensures a leader selection from the common prefix (R2). The resource allocator RA_{pos} uses $\mathbb{S}^{e-2}_{\text{stake}}$ together with the sampled randomness as input to the leader selection function F to check if p_i is selected for the slot sl. If this is the case, then RA_{pos} *assigns* the resource commitment to p_i, otherwise it *assigns* $\perp$. If the committed chain $\mathcal{C}$ is not valid, then RA_{pos} *assigns* $\perp$ to p_i. A validation of the resource commitment can be done by any process p_j through *RA-validate*; the resource allocator RA_{pos} returns

Algorithm 3 Implementing PoS Resource Allocator, RA_{pos}.

```
state
    $B_0$: Genesis block
    $\mathcal{D}$ : Distribution
    $F$ : Leader selection function
    $sl$ : Current slot, initially $sl = 0$
    $e$ : Current epoch, initially $e = 0$
    $k$ : common prefix parameter
    $q$ : number of slots in an epoch, initially $q = 16 \cdot k$
    $T$ : set of assigned resource commitments, initially empty
uses
    Random oracle: $H : \{0,1\}^* \to \{0,1\}^\lambda$
upon $RA\text{-}commit(p_i, st, r_i)$ s.t. $st = (\mathcal{C}, B)$, $B = (h, \overline{\mathsf{tx}}, \bot, \bot)$ do
    if $\mathcal{C}$ is valid $\wedge$ $H(\mathcal{C}[-1]) = h$ then
        obtain $\mathcal{C}_{\mathsf{prefix}}$ by pruning all blocks with slot $> (e-2) \cdot q$, from $\mathcal{C}$
        if $\mathcal{C}_{\mathsf{prefix}} = \emptyset$ do
            $\mathcal{C}_{\mathsf{prefix}} \leftarrow [B_0]$
        if $\exists (p_i, \mathcal{C}_{\mathsf{prefix}}, \rho^*, sl) \in T$ then          ▷ Queried before
            $\rho \leftarrow \rho^*$
        else
            $\rho \xleftarrow{R} \mathcal{D}$                              ▷ Sample a fresh randomness
            $T \leftarrow T \cup \{(p_i, \mathcal{C}_{\mathsf{prefix}}, \rho, sl)\}$                     ▷ Update $T$
        obtain the stake distribution $\mathbb{S}^{e-2}_{\mathsf{stake}}$ from $\mathcal{C}_{\mathsf{prefix}}$      ▷ Evaluate $StateAlloc(\cdot,\cdot)$
        $p_j \leftarrow F(\mathbb{S}^{e-2}_{\mathsf{stake}}, sl; \rho)$
        if $p_i = p_j$ then
            $\pi \leftarrow (p_i, \rho, sl)$
            output $RA\text{-}assign(p_i, st, \bot, \pi)$
        else
            output $RA\text{-}assign(p_i, st, \bot, \bot)$
    else
        output $RA\text{-}assign(p_i, st, \bot, \bot)$
upon $RA\text{-}validate(p_i, st, \pi)$ s.t. $st = (\mathcal{C}, B), B = (h, \overline{\mathsf{tx}}, \pi, \sigma), \pi = (p_j, \rho, sl)$ do
    obtain $\mathcal{C}_{\mathsf{prefix}}$ by pruning all blocks with slot $> (e-2) \cdot q$, from $\mathcal{C}$
    if $\mathcal{C}_{\mathsf{prefix}} = \emptyset$ do
        $\mathcal{C}_{\mathsf{prefix}} \leftarrow [B_0]$
    if $\exists (p_i, \mathcal{C}_{\mathsf{prefix}}, \rho, sl) \in T$ then                    ▷ Queried before
        obtain the stake distribution $\mathbb{S}^{e-2}_{\mathsf{stake}}$ from $\mathcal{C}_{\mathsf{prefix}}$
        $p_j^* \leftarrow F(\mathbb{S}^{e-2}_{\mathsf{stake}}, sl; \rho)$
        $b \leftarrow p_i \stackrel{?}{=} p_j^* \wedge \mathcal{C}$ is valid $\wedge$ $H(\mathcal{C}[-1]) = h$
        output $RA\text{-}is\text{-}committed(p_i, st, b)$
    else
        output $RA\text{-}is\text{-}committed(p_i, st, \text{FALSE})$
upon $Timeout$ do                                                ▷ Increment slot
    $sl \leftarrow sl + 1$
    if $sl \mod q = 0$ then                                       ▷ Increment epoch
        $e \leftarrow e + 1$
    $starttimer()$
```

either TRUE or FALSE, depending on the validity of the resource commitment. Finally, the PoS resource allocator is presented in Algorithm 3, and we conclude the following lemma and theorem.

▶ Remark 4.7. In practice, the randomness generation can be instantiated using verifiable random function [12], multiparty coin-tossing [20] protocol, or a random beacon [13]. However, Algorithm 3 aims to show the distinction between *external* and *virtual* resources.

▶ **Lemma 4.8.** *Given the random oracle $H(\cdot)$, the leader selection process ($\mathcal{D}$, F) parameterized by the default probability ϱ, and the network delay Δ, there exists a value $R_{\mathcal{A}}$ such that the resource allocator RA_{pos} implemented in Algorithm 3 is a secure resource allocator.*

Proof. *Liveness* property follows from the algorithm: upon $RA\text{-}commit(p_i, st, r)$ from process p_i, the resource allocator RA_{pos} either *(i)* has L86 satisfied and, eventually, outputs $RA\text{-}assign(p_i, st, \bot, \pi)$ with a resource commitment π to p_i, or outputs $RA\text{-}assign(p_i, st, \bot, \bot)$ to p_i (L100) or *(ii)* if the chain is invalid (L102), the allocator outputs $RA\text{-}assign(p_i, st, \bot, \bot)$ to p_i.

For the *validity* property, observe that $\pi = (p_i, \rho, sl)$ is valid if and only if p_i is a leader for sl. If p_i is a leader for sl, then in T there must be the random value ρ previously sampled for p_i (L94). This means that F evaluated on ρ will output again p_i. Hence, π can be validated by any process p_j through $RA\text{-}validate(p_j, st, \pi)$; RA_{pos} checks if $\pi \in T$ outputting the same result to p_j.

The *Use-once* property follows because, in our model, RA_{pos} keeps track of previous $RA\text{-}commit$ from p_i along with the time slots and states. Moreover, the choice of prob_i is stake-invariant, and it ensures that an adversary cannot increase its probability of being elected leader by dividing its stake into multiple identities. The proof for this is identical to the proof in Lemma 4.4. In practice, this property is enforced by the deterministic outputs of VRF and Hash function along with slot number and the common chain prefix as input.

The *unforgeability* property follows from the fact that any resource commitment produced by RA_{pos} is stored by the resource allocator in a set T of assigned resource commitments (L94). Hence, it is not possible for any process p_i to produce a valid resource commitment that is not in T. In practice, this property is guaranteed by the uniqueness property of verifiable random functions or the collision-resistant property of hash functions.

For the *honest-majority assignment* property, it is not difficult to see that we can derive ϱ_H and $\varrho_{\mathcal{A}}$ from R and $R_{\mathcal{A}}$. In particular, $\varrho_H = 1 - (1 - \varrho)^{R - R_A}$ and $\varrho_{\mathcal{A}} \approx 1 - (1 - \varrho)^{R_A}$. Here we note that the adversary can slightly increase $\varrho_{\mathcal{A}}$ by committing to shorter chains. However, it also means that the adversary will fall behind as it has to extend a much shorter chain than the current local chain maintained by correct processes, and we assume the adversary has no reason to do so. Hence, we consider $\varrho_{\mathcal{A}} = 1 - (1 - \varrho)^{R_A}$. Thus, we can derive R_A so that $\varrho_{\mathcal{A}} < \frac{1}{\Delta - 1 + 1/\varrho_H}$. ◀

▶ **Theorem 4.9.** *Algorithm 1 with the secure resource allocator RA_{pos} implements total-order broadcast.*

Proof. From Theorem 4.3 and Lemma 4.8, it follows that, since RA_{pos} is a secure resource allocator, then Algorithm 1 with RA_{pos} implements total-order broadcast. ◀

5 Trade-offs Between Different Resources

In this section, we describe various attacks against the resource-based total-order broadcast. In particular, we demonstrate long-range attacks against *virtual* resources, and we discuss the incentive consideration that describes the cost of launching attacks against *burnable* and *reusable* resources.

5.1 Virtual Resource vs External Resource: Long-Range Attacks

Long-range Attacks on Virtual Resources. Long-range attacks [10] (LRAs), also sometimes called posterior-corruption attacks, can be mounted on any blockchain based on a virtual resource (such as PoS) if the majority of the set of active processes from an earlier slot becomes inactive in a later slot, as they no longer have any stake left in the system. Formally, they can be defined as follows:

▶ **Definition 5.1** (Virtual-Resource-Shifting Event). *A* Virtual-Resource-Shifting Event *happens when there exist two values h_0, h_1, and a set of processes $\mathcal{P}_{\mathsf{maj}}$ such that:*

- ***Active at** h_0: At height h_0, processes in $\mathcal{P}_{\mathsf{maj}}$ control the majority of the total virtual resource (i.e., R), namely: $\sum_{p_i \in \mathcal{P}_{\mathsf{maj}}}(StateAlloc(p_i, \mathcal{C}[0:h_0])) > R - R_A$*
- ***Inactive at** h_1: At height $h_1 > h_0$, processes in $\mathcal{P}_{\mathsf{maj}}$ control less virtual resources than the total number of resources controlled by the adversary (i.e., $R_{\mathcal{A}}$): $\sum_{p_i \in \mathcal{P}_{\mathsf{maj}}}(StateAlloc(p_i, \mathcal{C}[0:h_1])) \leq R_{\mathcal{A}}$*

If Definition 5.1 is satisfied, then most processes in $\mathcal{P}_{\mathsf{maj}}$ have released all or part of their resources by height h_1, and the adversary has enough budget to corrupt all the active processes in $\mathcal{P}_{\mathsf{maj}}$ since they are all inactive in the present. The adversary could then use these processes to re-write the chain from $\mathcal{C}[h_0]$ since with a virtual resource as no external resource is needed to call the resource allocator. Furthermore, the *release of resource* from processes in $\mathcal{P}_{\mathsf{maj}}$ also happens on-chain, e.g., in the case of PoS for a process to move from active to inactive, it will spend its coins on-chain. An adversary re-writing the history of the chain could simply omit these transactions such that all processes satisfying definition 5.1 stay active in the alternative chain that the adversary is writing. The attack proceeds as follows:

1. When a virtual-resource-shifting event happens at the current height h_1, $\mathcal{A}$ corrupts all processes in $\mathcal{P}_{\mathsf{maj}}$. Since the total of resources controlled by these processes is less than $R_{\mathcal{A}}$, $\mathcal{A}$ has enough budget to do so;
2. $\mathcal{A}$ starts a new chain $\mathcal{C}^*$ at $\mathcal{C}[h_0]$. At this height, $\mathcal{A}$ controls the majority of the virtual resource, and because the resource allocator takes no further input apart from the state of $\mathcal{C}[0:h_0]$, it assigns the resource commitment to $\mathcal{A}$ with high probability;
3. $\mathcal{A}$ now controls all processes in $\mathcal{P}_{\mathsf{maj}}$ and can alter the state of the chain such that the processes in $\mathcal{P}_{\mathsf{maj}}$ never release their resource;
4. The adversarial chain will grow at a faster rate and will *eventually* become longer than the honest chain because there is no network delay between corrupted processes.

Long-range Attacks on External Resources. The strategy above does not work with *external* resources. Even if Definition 5.1 holds, the adversary cannot call the resource allocator by simply corrupting the processes p_i as an *external* resources would be needed as input to the resource allocator (step 2 in the strategy above).

We formalize the implication of the long-range attack in the following lemma and theorem.

▶ **Lemma 5.2** (Long-range Attack). *In a virtual-resource-based total-order broadcast (Algorithm 1), let $\mathcal{C}_{\mathsf{local}}$ be the longest chain maintained by a correct process, if a virtual-resource-shifting event occurs, then an adversary can* eventually *form a valid chain $\mathcal{C}^*$ that is longer than $\mathcal{C}_{\mathsf{local}}$.*

Proof. If a virtual-resource-shifting event (Definition 5.1) occurs, an adversary $\mathcal{A}$ can corrupt all $p_i \in \mathcal{P}_{\mathsf{maj}}$ at height h_1. Notice that $\mathcal{A}$ can do this because according to the threat model defined in definition 3.6, $\mathcal{A}$ has enough resource budget to corrupt all $p_i \in \mathcal{P}_{\mathsf{maj}}$.

Adversary $\mathcal{A}$ can start a new chain $\mathcal{C}^*$ at height h_0 by requiring all the corrupted processes $p_i \in \mathcal{P}_{\mathsf{maj}}$ to commit old states to RA_{pos}. Since the Byzantine processes control the majority of the resources, the probability of Byzantine processes getting assigned commitment is strictly higher than the probability of correct processes getting assigned commitment; hence, the growth rate of $\mathcal{C}^*$ is strictly higher than the growth rate of the honest chain $\mathcal{C}_{\mathsf{local}}$; therefore, $\mathcal{C}^*$ will eventually catch up and outgrow $\mathcal{C}_{\mathsf{local}}$ in terms of the length.

More concretely, to simplify our analysis, we also assume the network delay to be 1 (i.e., $\Delta = 1$) between correct processes. We recall that ϱ_H is the probability that at least one correct process gets selected on the honest chain $\mathcal{C}_{\mathsf{local}}$ at each time step. For any interval $[t_0, t_0 + t]$ and arbitrary $t_0, t \in \mathbb{N}$, we denote with $X_0, \ldots, X_{t-1}$ independent Poisson trails such that $\Pr[X_i = 1] = \varrho_H$, and we let $X_H = \sum_{i=0}^{t-1} X_i$. Using the Chernoff bound, one can show that for any $\epsilon \in (0,1)$ it holds that $\Pr[X_H < (1-\epsilon) \cdot \varrho_H \cdot t] \leq \exp(-\varrho_H \cdot t \cdot \epsilon^2/2)$. Intuitively, the Chernoff bound implies that the value of X_H cannot deviate too much from the mean; hence, for sufficiently large t and sufficiently small ϵ, the upper bound on the honest chain growth is approximately $\varrho_H \cdot t$, with an overwhelming probability.

Using the same argument for the growth of the malicious chain, one can show that for a sufficiently large time interval (i.e., t) and a sufficiently small ϵ, the lower bound of chain growth is approximately $\varrho_{\mathcal{A}} \cdot t$ (i.e., $(1+\epsilon) \cdot \varrho_{\mathcal{A}} \cdot t$) with an overwhelming probability (i.e., $\exp(-\varrho_{\mathcal{A}} \cdot t \cdot \epsilon^2/3)$), where $\varrho_{\mathcal{A}}$ is the probability that at least one Byzantine processes get selected on the honest chain $\mathcal{C}^*$ at each time step.

So, if $\varrho_{\mathcal{A}} > \varrho_H$, we can claim that $\mathcal{C}^*$ grows at a faster rate than $\mathcal{C}_{\mathsf{local}}$. This is the case for Algorithm 1 that uses RA_{pos} allocator. Due to Definition 5.1, the probability of getting assigned the resource commitment with $\mathcal{C}^*$, is $\varrho_{\mathcal{A}} > 1 - (1-\varrho)^{R - R_{\mathcal{A}}} = \varrho_H$, where ϱ_H is the probability that at least one correct processes get assigned a resource commitment with $\mathcal{C}_{\mathsf{local}}$. ◀

▶ Remark 5.3. Also, if we assume a $\Delta > 1$ network delay between correct processes, there will a non-zero probability that a fork can happen, and honest blocks can get discarded due to the network delay. On the other hand, we also assume a perfect synchrony ($\Delta = 1$) between Byzantine processes; therefore, there is no loss in the malicious growth rate. Therefore, even when correct processes and Byzantine processes control the same amount of resources on both chains, due to network delay, the chain growth rate of $\mathcal{C}^*$ can still be higher than the chain growth rate of the honest chain $\mathcal{C}_{\mathsf{local}}$.

▶ **Theorem 5.4.** *If a virtual-resource-shifting event occurs, a total-order broadcast based on virtual resources (Algorithm 1) does not implement total-order broadcast.*

Proof. Let $\mathcal{C}$ be the honest chain adopted by every correct process and let us assume that all the transactions buried at least k blocks deep in $\mathcal{C}$ have been *a-delivered* (Algorithm 1, L42) by every correct process. If a virtual-resource-shifting event occurs then, by Lemma 5.2, an adversary can eventually form a valid chain $\mathcal{C}^*$ that is longer than $\mathcal{C}$.

For existing processes, the adversary can send this $\mathcal{C}^*$ to a subset of correct processes. This implies that some correct processes will adopt $\mathcal{C}^*$ as a valid chain; they will *a-deliver* all the transactions buried at least k blocks deep in $\mathcal{C}^*$. This implies that, eventually, the *total-order* property is violated. Also, due to the permissionless nature of our model, correct processes might join the system at any time. Hence, new processes will adopt the malicious chain as the local chain; therefore, *delivered* will be different among correct processes. Hence, the *total-order* property is violated ◀

5.2 Incentives in Burnable and Reusable Resources

One of the vulnerabilities induced by *reusable* resources is that extending the blockchain is *costless* with respect to the resource considered. This is different from *burnable* resources, where creating a block consumes the resource; this consumption is captured in our model as the interaction between processes and the resource allocator. The use of reusable resources can result in two different types of adversarial behaviors. The first one consists in creating multiple blocks at the same time slots on different chains. The second one consists in keeping blocks created private from the rest of the processes. In both cases, we discuss how this *costless* property associated with block creation for longest-chain consensus protocols based on a reusable resource impacts their security compared to those based on burnable resource. In this section, we assume, as is traditional with any blockchain system, that some financial reward is associated with block creation, and we assume the cost of acquiring resources is the same for both reusable and burnable resources. With these assumptions and the use-once property of resource allocator, we define the chain extension cost as follows.

▶ **Definition 5.5** (Chain Extension Cost). *The cost of extending a valid chain for a process p_i between two time steps t_1 and t_2 such that $t_1 \leq t_2$ is defined to be the resource budgets committed and assigned back during this time interval. In particular, we have:*

- *For burnable resources:* $Cost_{\mathsf{burn}}(p_i, t_1, t_2) = \sum_{t=t_1}^{t_2} Alloc(p_i, t)$
- *For reusable resources:* $Cost_{\mathsf{reuse}}(p_i, t_1, t_2) \leq \max_{t \in [t_1, \ldots, t_2]}\{Alloc(p_i, t)\}$

▶ **Proposition 5.6.** *For all time step $t_2 > t_1$ and a process p_i, the cost of extending a valid chain with a burnable resource is strictly more expensive than with a reusable resource, i.e., $Cost_{\mathsf{burn}}(p_i, t_1, t_2) > Cost_{\mathsf{reuse}}(p_i, t_1, t_2)$.*

Proposition 5.6 indicates that it is inherently more expensive to extend the blockchain for *burnable* resources; hence, it is more difficult to launch different types of attacks on blockchain based on *burnable* resources. In the following, we explain different types of attacks.

Private Attack. The private attack [11], sometimes called double-spending attack, is the most simple attack in longest-chain blockchains. The adversary creates a private chain, i.e., it mines on its own without broadcasting its blocks to the other processes and without accepting the blocks from other processes. In particular, the adversary runs Algorithm 1, except that it does not broadcast its blocks until the end of the attack. This means that two chains grow in parallel: the adversarial one, that only the adversary is aware of, and the honest one. The adversary is aware of the honest chain but chooses not to contribute to it and it wins the attack if it creates a chain longer than the honest chain. In the case of a *burnable* resource, this attack has a cost as every block created consumes a resource. If the adversary wins the attack, then the cost is recovered as the adversary wins the reward associated with block creation. Otherwise, it loses the cost associated with all the resources consumed. In the case of a *reusable* resource, the only cost of the attack is the *opportunity cost*, i.e., the adversary takes the risk of potentially not earning the rewards associated with block creation if the attack fails but does not lose any resources. The attack in this case is then much cheaper than in the case of a burnable resource. The cost of a private attack is higher if the resource allocator is based on a burnable resource than if it is on a reusable resource, thus creating a stronger disincentivisation for an adversary. The results follow from the fact that for a reusable resource, the resource allocator can be invoked on the same resource several times. From Proposition 5.6, it is not difficult to see that the expected return on performing a private attack is higher for a *reusable* resources as the probability of winning the attack (i.e. producing a longer chain) is the same in both cases, but the cost is higher for a *burnable* resource.

Resource-bleeding Attack. Stake-bleeding attacks [17] were proposed in the context of PoS blockchains and work, informally, as follows. An adversary starts creating a private chain (i.e., it does not broadcast its blocks to the rest of the network) but, differently from the private attack described previously, the adversary may continue creating blocks on the honest chain. In its private chain, the adversary includes all of the transactions it is aware of, harvesting the associated transaction fees. Furthermore, the adversary also receives the coinbase reward usually associated with block production. After a sufficient amount of time, the adversary will have bloated its amount of resources and will *eventually* be able to create a chain that becomes as long as the honest chain. This attack could be extended to the general-resource case, which we call this attack *resource-bleeding attack*, and note that in the case of an external resource, this attack is much easier to detect than in the Proof-of-Stake case. In order to understand this attack, we must extend the model from Section 4 to take into account *total resource adjustments* in the case of inactive processes. In Section 6, we describe the general case of resource-bleeding attacks and discuss how they are more detectable on an external resource and the most mitigated for burnable resources.

Nothing-at-Stake Attack. In a Nothing-at-Stake attack, instead of deciding to extend the longest chain (Algorithm 1, L25), a process decides to mine simultaneously on all of the chains it is aware of. In the case of a burnable resource, an adversary cannot reuse the same resource to mine on multiple chains (due to L49), hence in order to mount this attack, the adversary must decide how to commit its resources to multiple chains. In contrast, with a reusable resource, each resource can be fully committed to each chain. If there exists multiple forks of the same length, there is a risk that a process will mine on a chain that ends up being abandoned and thus will miss out on the associated reward. It thus becomes rational for a process to deviate from the protocol and mine on every chain since this reduces chance of losing reward because network may select different chain. If every process adopts this strategy, the protocol cannot achieve the common prefix property as every chain will keep on growing at the same pace.

6 Discussion

Resource-bleeding Attack in the Flexible Resource Setting. The resource-bleeding attack stems from this observation: in order to deal with inactive processes, if the protocol wants to maintain its block production rate, it needs to adjust its leader selection processes such that inactive processes are not selected anymore. In practice, this means increasing ϱ such that every active process has a higher chance of being selected and removing the inactive processes from the list of eligible block producers and hence maintaining a steady block rate. If an adversary starts a private attack, since no resource commitment from the other processes is included in the adversarial chain, after a sufficient time, ϱ will be updated to ensure that the adversarial chain block rate is maintained. On the other hand, with a reusable resource, the adversary could keep maintaining its resources on the honest chain to ensure that the leader selection probability is not adjusted on the honest chain. After enough time, all the honest processes will be removed from the power table in the adversarial chain. This means that when electing a leader on the adversarial chain, the adversary now represents the full power table and is guaranteed to be elected at each epoch. On the other hand, since the adversary maintain its resource on the honest chain, without contributing as many blocks as it could. This means that, after some time, the honest chain will grow at a slower rate than the adversarial chain and the adversary will be able to create a chain as long as the honest chain, breaking the safety of the protocol.

In practice in Bitcoin, the *target* value [26] is updated every two weeks (roughly) to ensure that blocks are created, on average, at the same pace. An adversary could fork the chain, wait for the difficulty adjustment to adjust and then be able to create a chain at the same pace as the honest chain. This is, however, easily detectable. In the PoW case, one can simply see that the difficulty has been adjusted and that one chain has much fewer resources than the other. Moreover, since the resource is burnable, it is not possible for an adversary to continue mining on the honest chain as the same burnable resource cannot be used twice, hence the adversary cannot maintain its full resource on the honest chain and the honest chain difficulty must be adapted accordingly.

For an external, but reusable, resource such as storage, the adversary could maintain its power in both chain, however, it is easy to detect the adversarial chain as it will have fewer resources committed to it and hence is distinguishable from the honest chain.

Mitigations against Different Attacks. In the following, we discuss various mitigations against attacks described in Section 5.

Long-range Attacks. In practice, many PoS systems deal with long-range attacks by using some form of checkpointing [1, 24, 23], requiring key-evolving cryptography [9, 19], or using multiple types of resources [15]. Others use more refined chain selection rules [9, 2] (i.e., chain density analysis or selecting the longest chain that fork less than k blocks) instead of the longest chain selection.

Resource-bleeding Attacks. In the case of PoS, mitigation has been proposed in Ouroboros Genesis [2] and it works as follows. When a process is presented with two forks, it differentiates between two cases. In the first case, the fork is smaller than the common prefix parameter k, i.e., the two chains differ for a number of slots smaller than k, in which case the usual longest-chain rule is applied. If on the other hand, the forks differ from more than k slots, then the processes look at the first k slots after the fork (i.e., the first k slots where the two chains diverge) and choose the chain with the most blocks in that period. Intuitively, this is because during the beginning of the fork, an adversary has not had the time to bloat its stake and hence the rate at which its chain grows will be smaller than that of the correct processes. In the case of an external resource, it suffices to look at the total power (which can be explicit in the case of a reusable resource, or implicit for a burnable resource, e.g., *target* value) at the tip (end) of the chains and pick the one with the most resource.

Nothing-at-stake Attacks. A process that performs a nothing-at-stake attack with a reusable resource is easily detectable as anyone can see that the same resource was used on different chains. One typical mitigation adopted by PoS systems is to *slash*, i.e., financially punish, processes who use their resource on concurrent chains. This is usually done by having processes deposit some money before gaining participation rights, and then burning some of this deposit if a proof of misbehavior is sent to the blockchain. The details of this mechanism are out of scope for this paper.

7 Conclusion

Resources are essential in ensuring the safety property of total-order broadcast protocols in a permissionless setting as it protects the protocol from Sybil attacks. However, there exist several attacks on protocols based on reusable and virtual resources that a formal specification would help understand and address.

In this work, we formalize properties of resources through a *resource allocator* abstraction, and identify crucial properties on how to make this resource allocator secure for blockchain protocols. Using a secure resource allocator, we demonstrate how to construct a generic *longest-chain total-order* broadcast algorithm. Furthermore, we also illustrate how certain types of resources tend to make blockchain protocols more vulnerable to different types of attacks. We believe that this formalization will help blockchain protocol designers to select suitable types of resources for their protocols and understand and analyze the potential security trade-offs on those resources.

Outlook. For future work, we find the following research directions worth investigating:

- **Relaxed Assumptions.** Our analysis works with a setting where the total amount of active resources is known and fixed. Hence, it is natural to extend this model to a setting where the total amount of resources is unknown and potentially fluctuates.
- **Different Network Setting and Participation Models.** Our model focuses on *probabilistic longest-chain* protocols in a Δ-synchrony setting. However, we believe that our model can be applied to analyze properties of resource-based *deterministic* protocols in a permissioned and partially synchrony setting such as Tendermint [6] and HotStuff [27].
- **Different Types of Resources.** Finally, there are other resource-based protocols such as the Proof-of-Elapsed-Time (PoET) protocol [4] or multi-resources-based protocol [15] that have not been considered in this work. Hence, one can extend this model to analyze those protocols.

References

1 Sarah Azouvi, George Danezis, and Valeria Nikolaenko. Winkle: Foiling long-range attacks in proof-of-stake systems. In *AFT*, pages 189–201. ACM, 2020.

2 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.

3 Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. *IACR Cryptol. ePrint Arch.*, page 1015, 2015.

4 Mic Bowman, Debajyoti Das, Avradip Mandal, and Hart Montgomery. On elapsed time consensus protocols. In *INDOCRYPT*, volume 13143 of *Lecture Notes in Computer Science*, pages 559–583. Springer, 2021.

5 Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S. Matthew Weinberg. Formal barriers to longest-chain proof-of-stake protocols. In *EC*, pages 459–473. ACM, 2019.

6 Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018. `arXiv:1807.04938`.

7 Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.

8 Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography*, volume 11598 of *Lecture Notes in Computer Science*, pages 23–41. Springer, 2019.

9 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.

10 Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019.

11 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 *CCS*, pages 859–878. ACM, 2020.

12 Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 2005.

13 drand: Distributed randomness beacon. URL: `https://drand.love/`.

14 Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

15 Matthias Fitzi, Xuechao Wang, Sreeram Kannan, Aggelos Kiayias, Nikos Leonardos, Pramod Viswanath, and Gerui Wang. Minotaur: Multi-resource blockchain consensus. In *CCS*, pages 1095–1108. ACM, 2022.

16 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

17 Peter Gazi, Aggelos Kiayias, and Alexander Russell. Stake-bleeding attacks on proof-of-stake blockchains. In *CVCBT*, pages 85–92. IEEE, 2018.

18 Peter Gazi, Aggelos Kiayias, and Alexander Russell. Tight consistency bounds for bitcoin. In *CCS*, pages 819–838. ACM, 2020.

19 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68. ACM, 2017.

20 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388. Springer, 2017.

21 Andrew Lewis-Pye. Byzantine generals in the permissionless setting. *CoRR*, abs/2101.07095, 2021. `arXiv:2101.07095`.

22 Ling Ren. Analysis of nakamoto consensus. *IACR Cryptol. ePrint Arch.*, page 943, 2019.

23 Selma Steinhoff, Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolic. BMS: secure decentralized reconfiguration for blockchain and BFT systems. *CoRR*, abs/2109.03913, 2021. `arXiv:2109.03913`.

24 Ertem Nusret Tas, David Tse, Fangyu Gai, Sreeram Kannan, Mohammad Ali Maddah-Ali, and Fisher Yu. Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. *IACR Cryptol. ePrint Arch.*, page 932, 2022.

25 Benjamin Terner. Permissionless consensus in the resource model. In *Financial Cryptography*, volume 13411 of *Lecture Notes in Computer Science*, pages 577–593. Springer, 2022.

26 Bitcoin Wiki. Target. URL: `https://en.bitcoin.it/wiki/Target`.

27 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.