Multiparty Computation with Covert Security and Public Verifiability

Peter Scholl
Aarhus University, Denmark

Mark Simkin
Ethereum Foundation, Aarhus, Denmark

Luisa Siniscalchi
Aarhus University, Denmark
Concordium Blockchain Research Center, Aarhus, Denmark

Abstract

Multiparty computation protocols (MPC) are said to be *secure against covert adversaries* if the honest parties are guaranteed to detect any misbehavior by the malicious parties with a constant probability. Protocols that, upon detecting a cheating attempt, additionally allow the honest parties to compute certificates, which enable third parties to be convinced of the malicious behavior of the accused parties, are called *publicly verifiable*. In this work, we make several contributions to the domain of MPC with security against covert adversaries.

We identify a subtle flaw in a protocol of Goyal, Mohassel, and Smith (Eurocrypt 2008), meaning that their protocol does not allow to identify a cheating party, and show how to fix their original construction to obtain security against covert adversaries.

We present generic compilers that transform arbitrary passively secure preprocessing protocols, i.e. protocols where the parties have no private inputs, into protocols that are secure against covert adversaries and publicly verifiable. Using our compiler, we construct the first efficient variants of the BMR and the SPDZ protocols that are secure and publicly verifiable against a covert adversary that corrupts all but one party, and also construct variants with covert security and identifiable abort.

We observe that an existing impossibility result by Ishai, Ostrovsky, and Seyalioglu (TCC 2012) can be used to show that there exist certain functionalities that cannot be realized by parties, that have oracle-access to broadcast and arbitrary two-party functionalities, with information-theoretic security against a covert adversary.

2012 ACM Subject Classification Security and privacy → Cryptography

Keywords and phrases Multi-party computation, covert security, public verifiability

Digital Object Identifier 10.4230/LIPIcs.ITC.2022.8

Related Version *Full Version*: `https://eprint.iacr.org/2021/366.pdf`

Funding *Peter Scholl*: Supported by the Independent Research Fund Denmark (DFF) under project number 0165-00107B (C3PO) and Aarhus University Research Foundation.
Luisa Siniscalchi: Supported by the Concordium Blockchain Research Center.

1 Introduction

Secure multiparty computation (MPC) protocols allow collections of parties, where each party holds some private input, to compute arbitrary functions of those inputs in a way that reveals the result of the computation, but nothing else beyond that. Ideally, we would like our MPC protocols to be as fast and as secure as possible, but in reality we often have to choose one over the other. Protocols that are secure against passive adversaries who follow the protocol specification honestly, but try to learn more about the other parties' inputs from what they see, are typically quite fast, whereas protocols that are secure against actively misbehaving adversaries tend to be significantly slower.

To overcome the dilemma of needing to choose between good efficiency and good security, Aumann and Lindell [2] introduced an intermediate security notion which they call *security against covert adversaries*[1]. In a covertly secure protocol, corrupt parties may try to cheat and learn information about the honest parties' inputs, however, it is guaranteed that any such misbehavior will be detected with some constant probability ϵ, known as the *deterrence factor*. If cheating is detected then the honest parties will agree on at least one misbehaving party. The authors motivate covert security by arguing that, in certain scenarios, the repercussions of being caught misbehaving outweigh the gains that come from successfully cheating, so an adversary should be incentivized to behave honestly.

Subsequently, Asharov and Orlandi [1] proposed a strengthening of covert security, which requires the honest parties to not only detect misbehavior with a constant probability, but also prove it to third parties in a *publicly verifiable* manner. That is, the honest parties, upon detecting misbehavior during a protocol execution, should be able to compute a certificate that provably shows that at least one of the corrupted parties attempted to cheat.

Goyal, Mohassel and Smith [17] showed how to construct efficient two- and multiparty computation protocols with security against covert adversaries based on Yao's Garbled Circuits and its multiparty equivalent, the BMR protocol [6]. Damgård et al. [11] present a protocol in the preprocessing model, i.e. where the overall execution is split into an input-independent preprocessing and a input-dependent online phase, with a weaker notion of security against covert adversaries, where the misbehaving party is not necessarily identified, based on the SPDZ framework [14]. Damgård, Geisler, and Nielsen [10] present a compiler that transforms certain passively secure protocols based on secret sharing into protocols with security against covert adversaries. Their compiler only applies to protocols that assume an honest majority among the parties. None of the works above provide public verifiability.

The first two-party protocol with public verifiability was presented in the work of Asharov and Orlandi [1]. More efficient publicly verifiable two-party protocols with the same security guarantees have subsequently been proposed by Kolesnikov and Malozemoff [22] and Hong et al. [19]. In a recent work by Damgård, Orlandi, Simkin [13], the authors propose a generic compiler that transforms arbitrary two-party protocols with passive security into protocols with security against covert adversaries and public verifiability. The authors also sketch how to extend their compiler to the multiparty setting in the presence of an adversary that corrupts a constant fraction of the parties. Their multiparty protocols, however, have a deterrence factor that is inversely proportional to the fraction of corrupted parties and the resulting protocols are unlikely to be faster, in terms of concrete efficiency, than existing multiparty computation protocols with active security.

Given this state of the art, it is natural to ask whether we can construct MPC protocols, which provide security and public verifiability against an adversary that can corrupt all but one party, and are concretely more efficient than the fastest actively secure protocols.

[1] In the remainder of this paper, we will use "security against covert adversaries" and "covertly secure protocols" interchangeably.

1.1 Our Contribution

In this work, we make several contributions to the domain of MPC with security against covert adversaries with and without public verifiability.

1.1.1 On the Relation Between Covert and Active Adversaries

Firstly, we observe that in the multi-party setting (in contrast to two parties) there is a subtle but important difference between the standard definitions of covert security and active security used in the literature. The standard definition of covert security [2] explicitly requires that the honest parties agree upon the identity of a party who is caught cheating, a property we call *identifiable cheating*. Moreover, they require *identifiable abort*, meaning that a corrupt party who aborts the computation (without trying to learn additional information) is also identified. On the other hand, in the setting of dishonest majority, where more than half of the parties may be corrupted, actively secure protocols typically settle for security with abort, without identifiability (which is more expensive to realize). Hence, a covert secure protocol with identifiable abort is not strictly weaker than a standard actively secure protocol, but rather the two notions are incomparable. A more appropriate point of comparison is with an actively secure protocol with identifiable abort, which typically has a much higher cost [21]. Note that one can also consider a weaker notion of covert security with abort, where aborting parties are not identified, as has been done, for instance, in a covertly secure variant of the SPDZ protocol [11].

1.1.2 MPC with Security Against Covert Adversaries and Identifiable Abort

We identify a subtle flaw in the work of Goyal, Mohassel and Smith [17], which renders their multiparty protocol potentially insecure. More concretely, we show that while their solution correctly detects misbehavior with a constant probability, it does not necessarily allow the honest parties to unanimously agree on one of the misbehaving parties, so it does not satisfy the basic requirement of identifiable cheating. To fix their construction, we present a generic compiler for upgrading any passively secure preprocessing protocol, i.e. where the parties have no private inputs, into one with covert security and identifiable abort. This suffices to obtain a modified version of their construction with the desired security guarantees, namely, both identifiable cheating and identifiable abort. Furthermore, our compiler can be used to obtain a covertly secure variant of the SPDZ protocol with identifiable abort, improving upon the security of the covert protocol from [11], which does not have identifiable abort.

1.1.3 Preprocessing with Security and Public Verifiability Against Covert Adversaries

Our second compiler transforms any passively secure preprocessing protocol into a protocol with publicly verifiable covert (PVC) security (with the same corruption threshold). In the PVC setting, here we settle for security-with-abort and not identifiable abort. This is because publicly verifying that a party aborted may require proving to a third party that some corrupt party did not send a message; achieving this seems to require publishing the entire protocol transcript, which is a large extra cost.

Our compiler for PVC security leverages time-lock puzzles [23] in a novel fashion, to force a corrupt party to "commit" to opening some protocol executions before it learns whether or not a cheating attempt was successful. Importantly, the parties generate the puzzles locally, rather than inside MPC, and furthermore, the puzzles only have to be solved in case of a dispute, which allows us to construct concretely efficient protocols.

1.1.4 Applications

Our compilers for preprocessing apply to a large, general class of preprocessing functionalities for producing correlated randomness. To obtain a complete MPC protocol, we can apply one of our covert compilers to generate the preprocessing for an *actively secure* online phase. Even though we are then running an actively secure protocol, these are typically cheap and information-theoretic, with a much smaller overhead to achieve active security compared with the preprocessing phase. If the preprocessing has publicly verifiable covert security, then so does the combined protocol with the online phase, since any cheating in the online phase simply leads to abort. Similarly, if the preprocessing has covert security with identifiable abort, then so does the combined protocol as long as the online phase is secure with identifiable abort.

To illustrate this, we consider a few examples, namely the BMR [6], SPDZ [14] and TinyTable [12] families of protocols. These all perform MPC with up to $n - 1$ out of n corruptions and security with abort, for the settings of binary circuits, arithmetic circuits, and circuits augmented with "truth-table" gates, respectively. By applying our compiler, we can obtain the first preprocessing protocols for these, which achieve publicly verifiable covert security and improved performance. For example, to obtain a deterrence factor of $\epsilon = 2/3$, i.e. any misbehavior will be detected with probability $2/3$, our compiled covert protocol will, roughly, be only three times slower than its passively secure counterpart. Our compiler particularly shines for the TinyTable protocol [12], where we can improve performance by an order of magnitude or more when relaxing to covert security. This is because we are able to replace its costly, actively secure preprocessing with a much cheaper passive protocol. Also, as a contribution of independent interest, we present an optimized preprocessing phase for the passively secure BMR protocol, which reduces bandwidth in the preprocessing phase by around 20%, compared with previous methods [7, 9].

As mentioned earlier, a particularly interesting use-case is the setting of identifiable abort. Here, existing actively secure protocols based on SPDZ [4] and BMR [5] have a lot more overhead compared with the non-identifiable case, in particular because they require a secure broadcast channel. Our compiler is much simpler, and only needs broadcast in case some dishonest behaviour occurs, so we expect them to be much more efficient in typical usage.

1.1.5 Information-Theoretic Impossibility

Finally, we also show that there exist certain functionalities that cannot be realized with information-theoretic security against a covert adversary by parties that have oracle-access to a broadcast and arbitrary two-party functionalities. Our proof strategy for proving this impossibility is essentially identical to a previous proof by Ishai, Ostrovsky, and Seyalioglu [20], which shows that the same result holds if one aims for active security with identifiable abort. Similary to the actively secure identifiable abort compiler of [21], this motivates our approach of our covert security compilers, which make black-box access to the next-message function of a passively secure protocol, instead of general two-party functionalities like oblivious transfer. We do not claim any particular technical novelty here, but we provide the full proof in the full version for the sake of completeness.

1.1.6 Concurrent and Subsequent Work

Concurrently to our work,[2] Faust et al. [16] presented a compiler for publicly verifiable covert security. Their work does not consider identifiable abort, but is similar to our second compiler for public verifiability using time-lock puzzles. We mention here a few differences. Firstly, our protocol is more lightweight in its use of generic actively secure MPC, since we do not need to generate a time-lock puzzle inside MPC. This means we only invoke an actively secure protocol on a circuit with $O(nk\lambda)$ AND gates, for n parties, covert repetition factor k and security parameter λ, compared with $O(nk\lambda + \log^2 N)$ gates, where N is an RSA modulus, for the optimized variant of [16]. Secondly, in our protocol, less data is sent over a broadcast channel, since our compiler only broadcasts hashes of the underlying semi-honest protocol instead of the entire transcript. On the other hand, while our compiler requires the parties and judge to solve n time-lock puzzles, [16] only requires solving a single time-lock puzzle, and also uses the notion of a verifiable TLP, which leads to a more efficient judge, with a fast way of verifying puzzle solutions. Also, the security notions are slightly different, since [16] realizes a relaxed form of covert security, where the adversary may choose to cheat after learning its output, while we use the standard definition, but only consider relaxed preprocessing functionalities where corrupt parties may choose their own outputs (for preprocessing functionalities, this notion is implied by that of [16]). Finally, we give concretely efficient instantiations of our protocols, while [16] mainly analyzes asymptotic efficiency.

Subsequently to our work, the recent work of [15] introduces a new notion of financially backed covert security (FBC) which extends the notion of publicly verifiable covert security, where the corrupted party gets financially punished (rather than only being publicly detected). They generically transform different types of PVC protocols into FBC protocols, and their work can also be applied to our PVC protocols.

1.2 Technical Overview

1.2.1 Covert Security via Cut-and-Choose

Most existing protocols [2, 17, 10, 1, 11, 22, 19, 13] (with the exception of [13]) for secure computation with covert security follow the same general blueprint. They all start by considering some passively secure protocol that is run k times in parallel, where $k-1$ randomly chosen executions will eventually be opened and used for checking the behaviour of the involved parties and the last remaining execution will be used for computing the desired output. From a technical point of view, the main challenge is to find the right moment for revealing which executions are checked and opening them. If the executions are opened too early, then cheating may be possible in subsequent phases of these protocols. If they are opened too late, then there is a risk of revealing information about the private inputs of the honest parties.

A well-suited class of protocols, that implement some function f, to consider in this setting, are those that can be split into two phases: (1) a passively secure, but input-independent, preprocessing phase which realizes a correlated randomness functionality; (2) an actively secure online phase which takes as input the correlated randomness from the preprocessing phase in order to implement f. For technical reasons, we focus on passively secure preprocessing protocols which realize a mildly relaxed form of functionality called a

[2] Both papers were submitted to Eurocrypt 2021.

corruptible correlated randomness functionality [8]. Here, the functionality is parameterized by some distribution $\mathcal{D}$, and gives the adversary the power to choose the outputs from $\mathcal{D}$ that are given to the malicious parties; the functionality then *reverse samples* the output for the honest parties based on the malicious parties' output and the distribution $\mathcal{D}$. This class of functionalities encompasses those in popular MPC protocols like the BMR protocol [6] or the SPDZ framework [14]. Given such a protocol, we can run the preprocessing phase k times in parallel and check $k-1$ executions at the very end of that phase. If the check passes, the protocol proceeds to the actively secure online phase, where misbehavior is no longer a concern. Looking ahead, we would like to stress that our final protocols all satisfy the standard notion of covert security, despite using a relaxed preprocessing functionality.

Given this high-level blueprint, the remaining task is to design an appropriate check protocol that enables the honest parties to agree on a misbehaving party. To see whether some party P_i has sent the correct message in round r during the protocol execution, one needs to know P_i's private random tape and all messages P_i has received so far.

Based on these observations, a first attempt towards designing a check protocol could be as follows: Initially all parties commit to k random tapes each, i.e. each P_i for $i \in [n]$ commits to random tapes $r_{(i,1)}, \ldots, r_{(i,k)}$. The parties run the preprocessing protocol k times, where P_i uses random tape $r_{(i,j)}$ in execution j. Once all k executions terminate, the parties jointly flip a coin $c \in [k]$ and open all commitments via broadcast belonging to executions $j \in [k]$ with $j \neq c$. If any party aborts at this stage, we accuse it of cheating. If none of the parties abort, then everybody will know the vectors $(r_{(1,j)}, \ldots, r_{(n,j)})$ of random tapes used in executions $j \neq c$. Each party P_i can use the vectors of random tapes to generate a full honest transcript for each execution and use it to check whether it received messages that were consistent with the honest transcript during the protocol execution. Unfortunately, this approach has several problems.

The first one is that two honest parties, who may receive malformed messages from different adversarially corrupt parties during the protocol executions, have no obvious way of agreeing on which party to accuse unanimously. Even worse, a malicious party could falsely accuse some honest party to further make everyone's life more difficult.

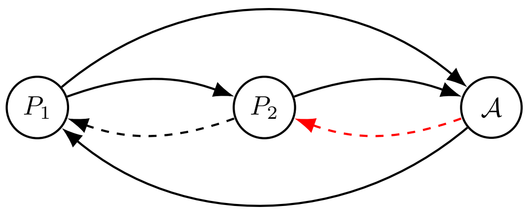

Figure 1 Illustration of inconsistent cheater identification.

The second, problem is that P_i receiving a message from P_j that is inconsistent with the honest transcript *does not* mean that P_j misbehaved. Consider the example in Figure 1, where all parties behave honestly (indicated by solid black lines) except for some corrupt party $\mathcal{A}$, which sends a malformed messages to P_2 (red dashed line).

Given that P_2's messages to P_1 may be a function of the messages it receives from $\mathcal{A}$, we potentially end up in a situation, where P_2 subsequently sends an incorrect, but honestly generated, message to P_1 (dashed black line). The takeaway from this discussion is that P_1 cannot simply compare the messages it received from other parties with the messages in an honest transcript to deduce who misbehaved in the protocol execution. This is exactly what goes wrong in the compiler of Goyal et al. [17], which tries to recover from cheating by opening the randomness for the underlying semi-honest protocol π. When π is instantiated using the GMW protocol based on pairwise OT channels, as suggested in [17], although cheating may be detected, it is not possible to identify the cheating party.

If the semi-honest protocol in their approach was adjusted to send every message over a broadcast channel (using public-key encryption), then their approach would be sound. This, however, would introduce an overhead of $\mathcal{O}(n)$ broadcasts; even when all parties behave honestly. Looking ahead, our protocol will only make use of broadcasts when malicious parties actively misbehave. Even in the presence of an adversary who tries to trigger as many broadcasts as possible, our protocol is still more efficient than the approach of Goyal et al., since for the specific case of BMR, we use various optimizations which means that only one instead of k garbled circuits needs to be broadcast. Additionally, it is worth noting that in the case of public verifiability covert security, we only need broadcast with abort, which can be instantiated much more efficiently.

1.2.2 Achieving Identifiable Abort

To get around the issue described above, we define a new property for MPC protocols that we call *identifiable cheating from random tapes (IDC)*. We say that a protocol has the given property, if there exist two protocols Certify and Identify associated with the given protocol. Certify takes the random tapes of all parties in an execution and the local view of some party P_i as input and outputs a partial certificate cert_i. The algorithm Identify takes n partial certificates as input and either outputs the index of a malicious party that misbehaved in the protocol execution or outputs $\perp$ to indicate that nobody misbehaved. Importantly, we require Identify to function correctly, even if the corrupted parties output false partial certificates or do not send anything. We show how to transform any passively secure protocol into one that supports IDC. The formal details and results regarding this property can be found in the full version. We remark that the notion of $\mathcal{P}$-verifiability from [21] achieves a similar goal, but this transformation (and the variant from [5]) uses broadcast so is less efficient.

Now, we can follow the blueprint for constructing covertly secure protocols outlined before, but instead of each party comparing its view to the messages in an honest execution, we use Certify and Identify to check whether any party misbehaved and if so which index to output. The details of this construction can be found in Section 3.

1.2.3 Public Verifiability

To obtain not only covert security, but also public verifiability, we have to overcome several additional challenges. The first problem is that the IDC property sketched above is not sufficient for producing *publicly* verifiable certificates. More concretely, the IDC property does not provide any guarantees about the output of Identify, when the adversary is additionally allowed to replace some of the honest parties' partial certificates with some maliciously chosen values. This could potentially enable the adversary to produce a collection of false partial certificates, which accuse an honest party of misbehaving. Hence, we need a stronger flavor of this building block, which we call *publicly verifiable identifiable cheating from random tapes (PVIDC)*, where Identify correctly identifies a malicious party or outputs $\perp$, even if the adversary is allowed to tamper with the honest parties' partial certificates. Here again, we show how to transform arbitrary passively secure protocols into ones that support PVIDC. Our transformation for obtaining PVIDC is slightly less efficient than the transformation for just IDC, but still significantly more efficient than running a fully actively secure protocol. The formal definition of this property can be found in the full version.

The second problem is, that upon revealing which executions should be checked, the adversary can simply stop responding and thereby prevent the honest parties from checking the executions and obtaining a certificate of the adversary's misbehavior. In contrast to covert security without public verifiability, here we cannot simply accuse the aborting party of cheating, because the honest parties have no corresponding publicly verifiable evidence. At first glance, our goals at this step may even seem contradictory. On one hand, during the checking phase, we would like to ensure that the adversary cannot tell whether the information it is about to reveal is useful for incriminating it. On the other hand, we would like to ensure that any other party can use the revealed information for determining whether cheating has happened or not and who the cheating party was.

To get out of this predicament, we make use of a tool called time-lock puzzles [23]. Such puzzles allow a sender to publish a message that cannot be read before a certain time has passed, e.g. in our case before at least some number of rounds in a synchronized network have passed. Crucially, the message becomes visible eventually, *without any interaction from the sender*. Time-lock puzzles can be built from RSA-based timing assumptions, without any trusted setup and generating a puzzle requires just a single exponentiation. Recently, time-lock puzzles have also been used to build 2-PC with output-independent abort [3], with a construction using similar ideas to ours (except that we are in the multi-party setting, and also achieve public verifiablity).

On an intuitive level, we use time-lock puzzles as follows:[3] At the beginning of the checking phase, all parties jointly execute an actively secure MPC protocol, where each party P_i inputs all k commitment openings that belong to the random tapes the party used. The MPC protocol picks $k-1$ executions at random and outputs time-lock puzzles of all random tapes belonging to those. Additionally, the parties obtain a secret sharing of the index c of the execution that is not being checked. All parties sign the time-lock puzzles, the commitments and broadcast the computed signatures. Because the puzzles cannot be solved fast enough, the adversary needs to decide whether to abort this phase of the execution without seeing the contents of the puzzles and thus without knowing which executions are being checked. Once the honest parties have signatures of the corrupted parties on the time-lock puzzles, they are, roughly speaking, guaranteed to have some useful information that can be shown to an external party in case cheating will be detected. Once all parties signed the time-lock puzzles, they all publish their share of c and then publish the openings of the random tapes used in the executions $j \neq c$. Now if the adversary decides to abort, because it doesn't like which executions are being checked, then the honest parties can obtain its necessary random tapes from the signed time-lock puzzles.

In our final protocol, the time-lock puzzles are generated locally by the parties, outside of MPC, and incur an overhead that is independent of the size of the circuit to be evaluated. Considering the evaluation of larger circuits, these additional costs from using time-lock puzzles become minor and in executions, where all parties behave honestly, no time-lock puzzles need to be solved by any party. The details of this protocol can be found in full version.

[3] We are omitting several important details here that can be found in the technical parts, e.g. in the full version of the paper.

1.2.4 Instantiating the Compilers

Our compilers can be instantiated with any MPC protocol in the preprocessing model, as long as its preprocessing functionality implements the *corruptible* correlated randomness functionality explained above. For most MPC protocols based on secret-sharing, this requirement is already satisfied out-of-the-box. This includes protocols such as SPDZ [14, 11], TinyTable [12] and a version of SPDZ with identifiable abort [4]. We therefore easily obtain covertly secure variants of these protocols (with public verifiability or identifiable abort) by plugging in a semi-honest version of the preprocessing, which improves efficiency by avoiding e.g. expensive zero-knowledge proofs typically used in SPDZ.

The case of constant-round MPC, based on garbled circuits, is slightly more challenging. Here, if we want public verifiability, we can again directly apply our compiler to a semi-honest version of the BMR protocol similar to [7, 18], since we observe this works with a corruptible preprocessing functionality (we in fact give an optimized semi-honest preprocessing protocol, which reduces the number of OTs by 25%).

For identifiable abort, however, we need to modify the BMR functionality so that (1) we get a secure online phase with identifiable abort, and (2) the BMR functionality should be a corruptible preprocessing functionality. We observe that (1) is straightforward to achieve, by having each party send a commitment to its share of the garbled circuit in the preprocessing protocol; this ensures that any party who sends an incorrect share in the online phase can by identified, and is also cheap to implement, since our compiler only needs this to be done with passive security.

However, this is not compatible with the definition of a corruptible preprocessing functionality, indeed we would have to reverse-sample an honest party's message and decommitment information, *after* the corresponding commitment is provided by the adversary. This strong form of equivocation is not possible with standard commitments. Instead, we use *unanimously identifiable commitments* [20], which can be built information-theoretically in such a way that allows this. Setting up these commitments involves a little more work in the preprocessing, but this overhead is independent of the circuit size, since the parties only commit to a hash of their garbled circuit shares.

2 Preliminaries

Notation

Let λ be the computational and δ be the statistical security parameter. We write $[n]$ to denote the set $\{1, \ldots, n\}$. For all algorithms that follow, we will regularly omit the security parameter input and it is understood that this input is provided implicitly. We define the view of a party in the execution of the protocol Π as the messages she received during an execution of Π along with her input and random tapes. For a functionality $\mathcal{F}$, we write $[\mathcal{F}]^{\mathsf{ida}}$ to denote the corresponding ideal functionality with identifiable abort.

Secure Broadcast

We assume a broadcast channel, as well as a public-key infrastructure (PKI) which is implied by broadcast. In our protocols with public verifiability, it is enough to have broadcast with abort, which can be implemented with a cheap, 2-round echo protocol. For our protocols with identifiable abort, however the broadcast must be identifiable, which involves a more expensive protocol with $O(n)$ rounds and digital signatures.

Secure Multiparty Computation

All of our security definitions follow the ideal/real simulation paradigm in the standalone model. Throughout this paper we will consider protocols that are executed over a synchronous network with static, rushing adversaries and we assume the existence of secure authenticated point-to-point channels between the parties. The MPC definitions can be found in the full version, in particular the standard notion of covert security and covert security with publicly verifiability are defined; we remark that in the notion of covert security, we explicitly require *identifiable abort*, meaning that the honest parties agree upon the identity of the party who caused an abort. On the other hand, covert security with public verifiability is *not* identifiable; the adversary just sends the abort command without an index.

The other useful definitions, for instance time-lock puzzle, can be found in the full version.

2.1 Corruptible Correlated Randomness Functionality

For our work we will consider a mild relaxation of correlated randomness functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ called a *corruptible correlated randomness* functionality [8]. $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ allows the parties in the corrupted set C to choose their correlated randomness $\{R'_i\}_{i\in C}$ and then $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ has to reverse sample based on $\{R'_i\}_{i\in C}$ the correlated randomness for the honest parties consistently with the distribution $\mathcal{D}$. We model this equipping the functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ with an efficient reverse sample algorithm RS. We note that $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ is nonetheless sufficient for all major overall protocols in the preprocessing model.

Functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$

The functionality interacts with parties $P_1, \ldots, P_n$. Let $C \subset [n]$ be the set of parties corrupted by the ideal world adversary S.

Upon receiving message $(\texttt{CorrRand}, \{R'_i\}_{i\in C})$ from S, the functionality samples $\{R_i\}_{i\in[n]\setminus C} \leftarrow \mathsf{RS}(\{R'_i\}_{i\in C}, \mathcal{D})$ and sends R_i to each P_i with $i \in [n] \setminus C$.

Figure 2 Functionality for corruptible correlated randomness.

▶ Remark 1. We note that our ideal functionality implicitly requires the adversary S to provide adversarial correlated randomness that can be part of a valid output of the functionality. This is not a restriction, since we will later on prove that any real-world adversarial attack can be translated into such a restricted ideal-world adversary.

3 Preprocessing with Identifiable Abort

In this section we are presenting a protocol $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$ which implements $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ with security against covert adversaries, who corrupts $n-1$ parties, and deterrence factor $\epsilon = 1 - \frac{1}{k}$.

$[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$ makes use of a preprocessing protocol Π_{corr} that has identifiable cheating from random tapes (IDC), and consistent identifiable abort. Roughly speaking, $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$ proceeds as follow. Initially each party commits to k random tapes that are a result of a coin-flip, i.e. each P_i for $i \in [n]$ commits to random tapes $r_{i,1}, \ldots, r_{i,k}$. The parties run the preprocessing protocol Π_{corr} k times, where P_i uses random tape $r_{i,\ell}$ in execution ℓ. Once all k executions terminate, the parties jointly flip a coin $c \in [k]$ and open all commitments via broadcast belonging to executions $\ell \in [k]$ with $\ell \neq c$. If any party aborts at this stage, we accuse it of cheating. If none of the parties abort, then everybody will know the vectors $(r_{1,\ell}, \ldots, r_{n,\ell})$

of random tapes used in executions $\ell \neq c$. Once each party P_i received vectors of random tapes she runs algorithms $\mathsf{Certify}$ and Ver of Π_{corr} in order to identify if a malicious party misbehaved. If no cheating is detected P_i outputs the output of the c-th execution of Π_{corr}.

The formal description of the protocol can be found in Figure 3. In the formal description to not overburden the notation we avoid to specify that when a party, say P_i, does not broadcast a message (or the parties receive an (abort, i) from the functionality) the parties terminates the computation sending $(\mathsf{corrupted}, i)$.

Protocol $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$covprot

Let Π_{corr} be a protocol that computes $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ with passive security and has identifiable cheating from random tapes and consistent identifiability abort.

1. Each party P_i receives a signing key sk_i and verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ from the PKI.
2. For $i, j \in [n]$, each P_j sends $(\mathsf{comFlip}, (i, 1)), \ldots, (\mathsf{comFlip}, (i, k))$ to $[\mathcal{F}_{\mathsf{FLIP}}]^{\mathsf{ida}}$.
3. For $i, j \in [n]$, each P_j sends $(\mathsf{openOne}, (i, 1), i), \ldots, (\mathsf{openOne}, (i, k), i)$ to $[\mathcal{F}_{\mathsf{FLIP}}]^{\mathsf{ida}}$.
4. For $i, j \in [n]$, each P_i receives $r_{i,1}, \ldots, r_{i,k}$ from the ideal functionality.
5. All parties jointly execute Π_{corr} in parallel k times, where party P_i uses random tape $r_{i,\ell}$ in the ℓ-th execution of the protocol. Let $R_{i,\ell}$ be the output of party P_i in execution ℓ.
6. All parties send $(\mathsf{coinFlip}, 0)$ to $[\mathcal{F}_{\mathsf{FLIP}}]^{\mathsf{ida}}$ and obtain a uniformly random value $\ell^* \in [k]$.
7. For $\ell \in [k] \setminus \{\ell^*\}$ each party P_i
   a. sends $(\mathsf{openAll}, (i, \ell))$ to $[\mathcal{F}_{\mathsf{FLIP}}]^{\mathsf{ida}}$.
   b. receives back $\vec{r}_\ell := (r_{1,\ell}, \ldots, r_{n,\ell})$ corresponding to the random tapes used in execution ℓ from $[\mathcal{F}_{\mathsf{FLIP}}]^{\mathsf{ida}}$.
   c. computes $\mathsf{cert}_{i,\ell} \leftarrow \mathsf{Certify}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, \vec{r}_\ell, \mathsf{view}_{i,\ell})$, where $\mathsf{view}_{i,\ell}$ is P_i's view in the ℓ-th execution of Π_{corr}.
   d. send $\mathsf{cert}_{(i,\ell)}$ to $[\mathcal{F}_{\mathsf{BC}}]^{\mathsf{ida}}$.
8. Each party P_i receives certificates $(\mathsf{cert}_{1,\ell}, \ldots, \mathsf{cert}_{n,\ell})$ for each execution $\ell \neq \ell^*$ and computes $v_\ell \leftarrow \mathsf{Ver}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, \vec{r}_\ell, \mathsf{cert}_{1,\ell}, \ldots, \mathsf{cert}_{n,\ell})$. Let J be the set of indices with $v_\ell \neq \perp$. If $J \neq \emptyset$, then P_i broadcast $(\mathsf{corrupted}, v_j)$ with the smallest j from J.
9. Each party P_i outputs R_{i,ℓ^*}.

Figure 3 Preprocessing protocol for MPC with covert security and identifiable abort.

▶ **Theorem 2.** *Suppose protocol Π_{corr} securely implements $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ with passive security, has identifiable cheating from random tapes, and consistent identifiability abort. Let $[\mathcal{F}_{\mathsf{FLIP}}]^{\mathsf{ida}}$ be the ideal committed coin flip functionality with identifiable abort. Let $[\mathcal{F}_{\mathsf{BC}}]^{\mathsf{ida}}$ be the broadcast functionality. Then $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$ implements $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ with security against covert adversaries, who corrupts $n-1$ parties, and deterrence factor $\epsilon = 1 - \frac{1}{k}$.*

The ideal functionality used in the theorem statement, the proof of Theorem 2 and the definition of the property of cheating identifiability from random tapes can be also find in the full version of the paper.

References

1 Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 681–698. Springer, Heidelberg, December 2012. doi:10.1007/978-3-642-34961-4_41.

2 Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 137–156. Springer, Heidelberg, February 2007. doi:10.1007/978-3-540-70936-7_8.

3 Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In Anne Canteaut and Francois-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part III*, Lecture Notes in Computer Science, pages 429–459. Springer, Heidelberg, October 2021. doi:10.1007/978-3-030-77883-5_15.

4 Carsten Baum, Emmanuela Orsini, and Peter Scholl. Efficient secure multiparty computation with identifiable abort. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 461–490. Springer, Heidelberg, October / November 2016. doi:10.1007/978-3-662-53641-4_18.

5 Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 562–592. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56880-1_20.

6 Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*, pages 503–513. ACM Press, May 1990. doi:10.1145/100216.100287.

7 Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 578–590. ACM Press, October 2016. doi:10.1145/2976749.2978347.

8 Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 489–518. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26954-8_16.

9 Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. Motion – a framework for mixed-protocol multi-party computation. Cryptology ePrint Archive, Report 2020/1137, 2020. URL: https://eprint.iacr.org/2020/1137.

10 Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 128–145. Springer, Heidelberg, February 2010. doi:10.1007/978-3-642-11799-2_9.

11 Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority – or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, September 2013. doi:10.1007/978-3-642-40203-6_1.

12 Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 167–187. Springer, Heidelberg, August 2017. `doi:10.1007/978-3-319-63688-7_6`.

13 Ivan Damgård, Claudio Orlandi, and Mark Simkin. Black-box transformations from passive to covert security with public verifiability. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 647–676. Springer, Heidelberg, August 2020. `doi:10.1007/978-3-030-56880-1_23`.

14 Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, Heidelberg, August 2012. `doi:10.1007/978-3-642-32009-5_38`.

15 Sebastian Faust, Carmit Hazay, David Kretzler, and Benjamin Schlosser. Financially backed covert security. Cryptology ePrint Archive, Report 2021/1652, 2021. URL: `https://ia.cr/2021/1652`.

16 Sebastian Faust, Carmit Hazay, David Kretzler, and Benjamin Schlosser. Generic compiler for publicly verifiable covert multi-party computation. In Anne Canteaut and Francois-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, Lecture Notes in Computer Science, pages 782–811. Springer, Heidelberg, October 2021. `doi:10.1007/978-3-030-77886-6_27`.

17 Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 289–306. Springer, Heidelberg, April 2008. `doi:10.1007/978-3-540-78967-3_17`.

18 Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 598–628. Springer, Heidelberg, December 2017. `doi:10.1007/978-3-319-70694-8_21`.

19 Cheng Hong, Jonathan Katz, Vladimir Kolesnikov, Wen-jie Lu, and Xiao Wang. Covert security with public verifiability: Faster, leaner, and simpler. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 97–121. Springer, Heidelberg, May 2019. `doi:10.1007/978-3-030-17659-4_4`.

20 Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 21–38. Springer, Heidelberg, March 2012. `doi:10.1007/978-3-642-28914-9_2`.

21 Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 369–386. Springer, Heidelberg, August 2014. `doi:10.1007/978-3-662-44381-1_21`.

22 Vladimir Kolesnikov and Alex J. Malozemoff. Public verifiability in the covert model (almost) for free. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 210–235. Springer, Heidelberg, November / December 2015. `doi:10.1007/978-3-662-48800-3_9`.

23 Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto, 1996.