Multiparty Computation with Covert Security and Public Verifiability

Peter Scholl, Mark Simkin, and Luisa Siniscalchi

Aarhus University, 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) and show how to modify 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.

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 chose 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 [3] introduced an intermediate security no-

tion which they call *security against covert adversaries*[1]. A protocol is said to satisfy this notion, if it ensures that the honest parties detect any misbehavior by the adversarial parties with some constant probability ϵ, known as the *deterrence factor*, and allows them to agree on at least one misbehaving party. Formally, it should be noted that even though cheating is now possible with some non-negligible probability, the security notion is not strictly weaker than active security abort, since we require the protocol to identify the misbehaving party even if it just aborts the execution. The authors motivated their new security notion by arguing that, in certain scenarios, the repercussions of being caught misbehaving, outweigh the gains that come from successfully cheating and thus an adversary would be incentivized to behave honestly.

Subsequently, Asharov and Orlandi [2] proposed a strengthening of this security notion, which requires the honest parties to not only be able to detect misbehavior with a constant probability, but to also be able to 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 [22] 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 [7]. Damgård et al. [15] present a protocol in the preprocessing model, i.e. where the overall execution is split into a 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 [18]. Damgård, Geisler, and Nielsen [14] 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 [2]. More efficient publicly two-party protocols with the same security guarantees have subsequently been proposed by Kolesnikov and Malozemoff [28] and Hong et al. [24]. In a recent work by Damgård, Orlandi, Simkin [17], 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.

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

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.

1.1 Our Contribution

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

On the Relation Between Covert and Active Adversaries. Firstly, we observe that in the multi-party setting (in contrast with 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 [3] 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, actively secure protocols typically settle for security with abort, without identifiability. Hence, a covert secure protocol with identifiable abort is not necessarily weaker than a standard actively secure protocol. A more appropriate point of comparison is with an actively secure protocol with identifiable abort, which typically has a much higher cost [27].

MPC with Security Against Covert Adversaries and Identifiable Abort. We identify a subtle flaw in the work of Goyal, Mohassel, and Smith [22], 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 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. Our compiler can also be used to obtain a covertly secure variant of the SPDZ protocol with identifiable abort, which a previous construction with covert security [15] does not satisfy.

Preprocessing with Security and Public Verifiability Against Covert Adversaries. We present a second compiler, which transforms arbitrary passively secure preprocessing protocols into protocols with security and public verifiability against a covert adversary (with the same corruption threshold). Towards this goal, our compiler leverages time-lock puzzles [29] 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.

Applications. Our compilers are fully general, but we consider a few concrete applications, including the BMR [7], SPDZ [18] and TinyTable [16] families of protocols. These all perform MPC with up to $n-1$ out of n corruptions, 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 covert security with public verifiability. Since the actively secure variants of these are all significantly more expensive than covert, we obtain improved performance when switching to covert security. 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. 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 [8, 12].

As already mentioned above, a particularly interesting use-case is the setting of identifiable abort. Here, existing actively secure protocols based on SPDZ [5] and BMR [6] have a lot more overhead compared with the non-identifiable case, in particular because they require a secure broadcast channel. Our compiler are much simpler, and only need broadcast in case cheating occurs, so we expect them to be much more efficient in typical usage.

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 [26], 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 [27], this motivates our approach of obtaining covert security using black-box access to the next-message function of a passively secure protocol, instead of general two-party functionalities. We do not claim any particular technical novelty here, but we provide the full proof in Appendix A for the sake of completeness.

Concurrent work. Recently and concurrently[2] to our work, Faust et al. [20] presented a compiler for covert security with public verifiability. Their compiler is similar to our second compiler, although there are some key differences. Firstly, our compiler is more efficient than [20], since (1) we do not need to evaluate time-lock puzzles inside actively secure MPC (instead, we only use active MPC

[2] Both papers were submitted to Eurocrypt 2021.

for a much simpler building block), and (2) unlike [20], we do not require every message in the underlying passively secure protocol to be broadcast, which can increase costs by a factor $\approx n$. Secondly, the security notions are slightly different, since [20] realizes a relaxed form of covert security, where the adversary may choose to cheat after learning its output of the preprocessing functionality; on the other hand, we use the standard definition of covert security, but only consider relaxed preprocessing functionalities where corrupt parties may choose their own outputs. We also note that [20] does not consider convert security with identifiable abort, which our first compiler achieves. Finally, we show how to efficiently instantiate all of our protocols, which they do not.

1.2 Technical Overview

Covert Security via Cut-and-Choose. All of the existing protocols [3, 22, 14, 2, 15, 28, 24, 17] 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.

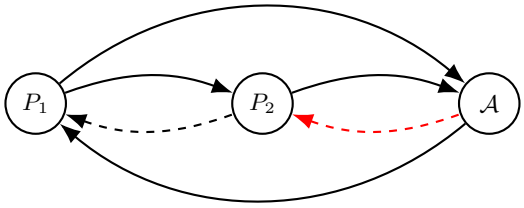
A well-suited class of protocols, that implement some function f, to consider in this setting, are those that can be split into two phases: (1) a passively secure, but input-independent, preprocessing phase which realizes a correlated randomness functionality; (2) an actively secure online phase which 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 realise a mildly relaxed form of functionality called a *corruptible correlated randomness* functionality [11]. 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 [7] or the SPDZ framework [18]. 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.

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 at 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.

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 1.2, 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. [22], 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 [22], 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 and even in the presence of an adversary that tries to trigger as many broadcasts as possible, our protocol will remain more efficient that the approach of Goyal et al.

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 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 Section 3.1. We remark that the notion of $\mathcal{P}$-verifiability from [27] achieves a similar goal, but this transformation (and the variant from [6]) 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 4.

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 certificate, which accuse an honest party of misbehaving. Hence, we need a stronger flavor of this building block, which we call *publicly verifiable cheating from random tapes (PVC)*, 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 PVC. Our transformation for obtaining PVC 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 Section 3.4.

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 that called time-lock puzzles [29]. 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 [4], 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.

[3] We are omitting several important details here that can be found in the technical parts, e.g. Section 5, of the paper.

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 Section 5.

Instantiating the Compilers (Section 6). 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 [18, 15], TinyTable [16] and a version of SPDZ with identifiable abort [5]. 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 [8, 23], 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* [26], 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 parties in the execution of the protocol Π as the messages she received during an execution of Π along with his input and random tapes. In this paper we are assuming broadcast and public-key infrastructure (PKI) which is implied by broadcast. For a functionality $\mathcal{F}$, we write $[\mathcal{F}]^{\mathsf{ida}}$ to denote the corresponding ideal functionality with identifiable abort.

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.

Covert adversaries [3]. The security notion we consider here is the strongest one of several and is known as the Strong Explicit Cheat Formulation (SECF). Covert adversaries are modeled by considering active adversaries, but relaxing the ideal functionality we aim to implement. The relaxed ideal functionality $\mathcal{F}_{\mathsf{SECF}}$ allows the ideal-world adversary S to perform a limited amount of cheating. That is, the ideal-world adversary, can attempt to cheat by sending cheat to the ideal functionality, which randomly decides whether the attempt was successful or not. With probability ϵ, known as the *deterrence factor*, $\mathcal{F}_{\mathsf{SECF}}$ will send back $\mathsf{detected}$ and all parties will be informed of at least one corrupt party that attempted to cheat. With probability $1-\epsilon$, the simulator S will receive $\mathsf{undetected}$. In this case S learns all parties' inputs and can decide what the output of the ideal functionality is. The ideal execution proceeds as follows:

Inputs: Every honest party P_i sends its inputs x_i to $\mathcal{F}_{\mathsf{SECF}}$. The ideal world adversary S gets auxiliary input z and sends inputs on behalf of all corrupted parties. Let $\bar{x} = (x_1, \ldots, x_n)$ be the vector of inputs that the ideal functionality receives.
Abort options: If a corrupted party sends (abort, i) (where party i is corrupted) as its input to the $\mathcal{F}_{\mathsf{SECF}}$, then the ideal functionality sends (abort, i) to all honest parties and halts. If a corrupted party sends $(\mathsf{corrupted}, i)$ as its input, then the functionality sends $(\mathsf{corrupted}, i)$ to all honest parties and halts. If multiple corrupted parties send (abort, i), respectively $(\mathsf{corrupted}, i)$, then the ideal functionality only relates to one of them. If both $(\mathsf{corrupted}, i)$ and (abort, i) messages are sent, then the ideal functionality ignores the $(\mathsf{corrupted}, i)$ messages.
Attempted cheat: If S sends (cheat, i) as the input of a corrupted P_i, then $\mathcal{F}_{\mathsf{SECF}}$ decides randomly whether cheating was detected or not:

- $\mathsf{Detected}$: With probability ϵ, $\mathcal{F}_{\mathsf{SECF}}$ sends $(\mathsf{corrupted}, i)$ to the adversary and all honest parties.
- $\mathsf{Undetected}$: With probability $1-\epsilon$, $\mathcal{F}_{\mathsf{SECF}}$ sends $\mathsf{undetected}$ to the adversary. In this case S obtains the inputs $(x_1, \ldots, x_n)$ of all honest parties from $\mathcal{F}_{\mathsf{SECF}}$. It specifies an output y_i for each honest P_i and $\mathcal{F}_{\mathsf{SECF}}$ outputs y_i to P_i.

The ideal execution ends at this point. If no corrupted party sent (abort, i), $(\mathsf{corrupted}, i)$ or (cheat, i), then the ideal execution continues below.
Ideal functionality answers adversary: The ideal functionality computes $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and sends it to S.
Ideal functionality answers honest parties: The adversary S either sends back $\mathsf{continue}$ or $(\mathsf{abort}, \mathsf{i})$ for a corrupted P_i. If the adversary sends $\mathsf{continue}$, then the ideal functionality returns y_i to each honest parties P_i. If the adversary sends $(\mathsf{abort}, \mathsf{i})$ for some i, then the ideal functionality sends back $(\mathsf{abort}, \mathsf{i})$ to all honest parties.
Output generation: An honest party always outputs the message it obtained from $\mathcal{F}_{\mathsf{SECF}}$. The corrupted parties output nothing. The adversary outputs an arbitrary probabilistic polynomial-time computable function of the initial inputs of the corrupted parties, the auxiliary input z, and the messages received from the ideal functionality.

The outputs of the honest parties and S in an ideal execution is denoted by $\mathsf{IDEAL}^{\epsilon}_{\lambda}[S(z), I, \mathcal{F}_{\mathsf{SECF}}, \bar{x}]$. Note that the definition requires the adversary to *either* cheat or send the corrupted parties' inputs to the ideal functionality, but not both.

Definition 1. *Protocol Π is said to securely compute $\mathcal{F}$ with security against covert adversaries with ϵ-deterrent in the $\mathcal{G}$-hybrid model if for every non-uniform probabilistic polynomial time adversary $\mathcal{A}$ in the real world, there exists a probabilistic polynomial time adversary S in the ideal world such that for all $\lambda \in \mathbb{N}$*

$$\left\{\mathsf{IDEAL}^{\epsilon}_{\lambda}[S(z), I, \mathcal{F}_{\mathsf{SECF}}, \bar{x}]\right\}_{\bar{x}, z \in \{0,1\}^*} \equiv_c \left\{\mathsf{REAL}_{\lambda}[\mathcal{A}(z), I, \Pi, \mathcal{G}, \bar{x}]\right\}_{\bar{x}, z \in \{0,1\}^*}$$

Remark 1. The notion of covert security, as defined above, explicitly requires identifiable abort*, meaning that the honest parties agree upon the identity of the party who caused an abort. We also consider a weaker definition where in case of abort, the adversary just sends the* abort *command without specifying any index, i.e.,* covert security with abort*.*

Security against covert adversaries with public verifiability. This notion was first introduced by [2] and was later simplified by [24]. In covert security with public verifiability, each protocol Π is extended with an additional algorithm $\mathtt{Judge}$. We assume that whenever a party detects cheating during an execution of Π, it outputs a special message cert. The verification algorithm, $\mathtt{Judge}$, takes as input a certificate cert and outputs the identity, which is defined by the corresponding public key, of the party to blame or $\perp$ in the case of an invalid certificate.

With public verifiability, we relax the abort option to have standard abort instead of identifiable abort. This relaxation makes sense because it does not seem possible to prove to a judge that a party aborted a computation, unless the judge has access to the entire transcript of the protocol (so it can tell, e.g., that some party stopped responding), and we do not wish to require this.

Definition 2 (Covert security with ϵ-deterrent and public verifiability). *Let $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ be the keys of the parties and f be a public function. We say that $(\Pi, \mathtt{Judge})$ securely computes f in the presence of a covert adversary with ϵ-deterrent and public verifiability if the following conditions hold:*

Covert Security: *The protocol Π (which now might output* cert *if an honest party detects cheating) is secure against a covert adversary with ϵ-deterrent factor according to the strong explicit cheat formulation described above, but with a difference in the abort options which is as follow: if a corrupted party sends* abort *to $\mathcal{F}_{\mathsf{SECF}}$ the functionalities sends* abort *to the honest parties and halts.*
Public Verifiability: *If the honest party $P \in [n]$ outputs* cert *in an execution of the protocol, then $\mathtt{Judge}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, f, \mathsf{cert}) = \mathsf{pk}_{\{1,\ldots,n\}\setminus P}$*[4] *except with negligible probability.*
Defamation-Freeness: *If the set of honest party $\mathbb{P}$ runs the protocol with corrupt parties $\mathbb{A}$ controlled by $\mathcal{A}$, then the probability that $\mathcal{A}$ outputs cert^* such that $\mathtt{Judge}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, f, \mathsf{cert}^*) = \mathsf{pk}_i$ with $i \in \mathbb{P}$ is negligible.*

The additional MPC definitions can be found in Appendix B.

2.1 Time-Lock Puzzles

Definition 3. *Let $B : \mathbb{N} \leftarrow \mathbb{N}$ and $\in (0,1)$. Let $\mathsf{TLP} = (\mathsf{pGen}, \mathsf{pSol})$ be a B-secure time-lock puzzle with the following syntax:*

- $z \leftarrow \mathsf{pGen}(1^\lambda, t, s)$: *A PPT algorithm that on input a security parameter $\alpha \in \mathbb{N}$, a difficulty parameter $t \in \mathbb{N}$, and a solution $s \in \{0,1\}^\lambda$, outputs a puzzle $z \in \{0,1\}^\lambda$.*
- $s = \mathsf{pSol}(1^\lambda, t, z)$: *A deterministic algorithm that on input a security parameter $\alpha \in \mathbb{N}$, a difficulty parameter $t \in \mathbb{N}$, and a puzzle $z \in \{0,1\}^\lambda$ outputs a solution $s \in (\{0,1\}^\lambda \cup \{\perp\})$.*

We require $(\mathsf{pGen}, \mathsf{pSol})$ to satisfy the following properties.

- **Correctness:** *For every λ, $t \in \mathbb{N}$, solution $s \in \{0,1\}^\lambda$, and $z \in \mathsf{SupppGen}(1, t, s)$, it holds that $\mathsf{pSol}(1^\lambda, t, z) = s$.*
- **Efficiency:** *There exist a polynomial p such that for all λ, $t \in \mathbb{N}$, $\mathsf{pSol}(1^\lambda, t, \cdot)$ is computable in time $t \cdot p(\lambda, \log t)$.*
- **B-Hardness:** *For sufficiently large λ, any pair of solutions $s_0, s_1 \leftarrow \{0,1\}^\lambda$, for a uniformly random bit $b \leftarrow \{0,1\}$, any PPT adversary $\mathcal{A}$ that gets $z \leftarrow \mathsf{pGen}(1^\lambda, B(\lambda), s_b)$ as input in round i and outputs b' in round $i + \ell$ for $1 \le \ell \le B(\lambda)$, it holds that $\Pr[b = b'] \le \frac{1}{2} + \mathsf{negl}(\lambda)$.*

Remark 2. Note that in the usual definitions for time-lock puzzles [19, 9], the hardness property is defined with respect to the depth of a circuit that attempts to solve the puzzle. Hardness in that definition states that circuits of a certain

[4] In the rest of the paper we are assuming that the $\mathtt{Judge}$ is already equipped with f.

bounded depth cannot solve the puzzle. Our definition is equivalent to saying that, in a synchronised network, all parties can only execute computations of a certain depth in one round.

The other useful definitions can be found in Appendix C.

2.2 Corruptible Correlated Randomness Functionality

For our work we will consider a mild relaxation of correlated randomness functionality $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ called a *corruptible correlated randomness* functionality [11]. $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ allows the parties in the corrupted set C to chose their correlated randomness $\{R'_i\}_{i\in C}$ and then $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ 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 $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ with an efficient reverse sample algorithm RS. We note that $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ is nonetheless sufficient for all major overall protocols in the preprocessing model as described in Section 6.

Figure 2.1: Functionality $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$

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 $i \in [n] \setminus C$.

Remark 3. 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 Public Verifiable Cheating and Identifiable Cheating from Random Tapes

In this section, we define two important properties on top of a passively secure protocol, which are: *identifiable cheating from random tapes* and *publicly verifiable cheating from random tapes*. Intuitively, these allow a party (or third party) to identify someone who misbehaved in the protocol, when they are given all parties' random tapes, and an additional, short certificate from the other parties. We give simple transformations for obtaining these given any passively secure protocol for a preprocessing functionality. Later, in Sections 4 and 5, we use these transformed protocols to build our covert secure preprocessing protocols with identifiable abort, and public verifiability, respectively.

3.1 Identifiable Cheating from Random Tapes

Below we define a notion of *consistent identifiability* for a protocol. This is similar to the standard definition of identifiable abort, where we require that in case of abort, the honest parties agree upon the identity of a corrupted party; however, consistent identifiability does not require any further security properties, so is independent of the protocol being passive or active secure.

Definition 4 (Consistent identifiability). *Protocol Π has* consistent identifiability*, if for any active, p.p.t. adversary corrupting a set of parties $A \subset [n]$ in an execution of Π, with overwhelming probability it holds that if any honest party outputs abort_i, then* all *honest parties output abort_i, and also $i \in A$.*

Before presenting our notion of verifiability, we need to specify what it means for a party to cheat in an execution of a protocol. We do not care about the case where malicious parties send incorrect messages to each other, so we say that cheating happens whenever a corrupted party sends a message to an honest party that is inconsistent with its random tape.

Definition 5 (Dishonest execution). *Consider a non-aborting execution of protocol Π between parties $P_1, \ldots, P_n$ with random tapes $(r_1, \ldots, r_n)$, and a set A of corrupted parties. We say that the execution was* dishonest with respect to *A, if there exists at least one honest party whose view in the execution is different to the view of the same party in an honest execution of Π on $(r_1, \ldots, r_n)$.*

We consider protocols with a simple kind of public-key infrastructure setup, where each party has a signing key sk_i, and access to the public verification keys vk_j of all other parties.

Definition 6 (Identifiable cheating from random tapes). *Let Π be a protocol between parties $P_1, \ldots, P_n$ in the PKI model, that takes no inputs. Suppose there are two deterministic polynomial-time algorithms:*

- $\mathsf{Certify}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{view}_i, \mathsf{sk}_i)$*: On input all the public keys vk_j and random tapes r_j, plus a view view_i and secret key sk_i of some party P_i from an execution of Π, this outputs a partial certificate $\mathsf{cert}_i \in \{0,1\}^*$.*
- $\mathsf{Identify}(\mathsf{vk}_1, r_1, \mathsf{cert}_1, \ldots, \mathsf{vk}_n, r_n, \mathsf{cert}_n)$*: On input all parties' public keys, random tapes and partial certificates, this outputs either the identity of a corrupted party, P_j, or an honest execution symbol $\perp$.*

Π supports identifiable cheating from random tapes (IDC) *if for any p.p.t. adversary $\mathcal{A}$ it holds that:*

$$\Pr[\mathtt{Exp}^{\mathsf{idc}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \nu(\lambda)$$

where $\lambda \in \mathbb{N}$, ν is a negligible function and $\mathtt{Exp}^{\mathsf{idc}}_{\mathcal{A},\Pi}(\lambda)$ is defined as follows:

Figure 3.1: Experiment $\mathtt{Exp}^{\mathsf{idc}}_{\mathcal{A},\Pi}(\lambda, x)$

1. *$\mathcal{A}$ corrupts a set of parties $A \subset [n]$.*
2. *For each $i \in [n]$, sample a random tape r_i and a signing key pair $\mathsf{vk}_i, \mathsf{sk}_i$*
3. *The parties run Π, where party P_i is given r_i, sk_i and $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ as input. Let view_i denote the list of messages received by P_i.*
4. *If there is an honest party with output abort_j for some j, output 0. Otherwise, continue.*
5. *Give to $\mathcal{A}$ the partial certificates $\mathsf{cert}_i = \mathsf{Certify}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{view}_i)$ and r_i, for $i \in [n] \setminus A$.*
6. *$\mathcal{A}$ outputs cert_j, for all $j \in A$.*
7. *Output 1 if one of the following holds:*
   (a) The execution of Π is dishonest with respect to A, and $\mathsf{Identify}((\mathsf{vk}_i, r_i, \mathsf{cert}_i)_{i=1}^n) = \perp$; or
   (b) $\mathsf{Identify}((\mathsf{vk}_i, r_i, \mathsf{cert}_i)_{i=1}^n) \in \{P_i\}_{i \notin A}$

   Otherwise, output 0.

3.2 Compiler for Identifiable Cheating

To construct a protocol with identifiable cheating from random tapes, we require the following basic property of the underlying passive protocol, which says that even if some parties misbehave in the protocol, the honest parties will always output some valid string. Note that this does not give any guarantee on the *correctness* of the outputs, we simply require they all output something, rather than for instance, a special abort symbol.

Definition 7. *We say that protocol Π is* well-defined with respect to malicious behaviour*, if in an execution with an actively corrupted set of parties, the protocol is guaranteed to terminate and every honest party outputs some value in $\{0,1\}^*$.*

Observe that it is easy to transform any passive protocol to be well-defined with respect to malicious behaviour, by just having the parties output a default value if another party either stops responding or sends an invalid message (such as of the wrong length).

Compiler. Our compiler takes any passive preprocessing protocol that is well-defined with respect to malicious behaviour, and converts it into a protocol in the PKI model that has passive security with consistent identifiability and supports identifiable cheating from random tapes.

We assume that we start with a protocol Π, in which the parties use pairwise communication channels to securely compute some preprocessing functionality $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$. Let NMF be the next message function for Π, that on input a party index i, round number ρ, random tape r_i and history of previously received messages $\mathcal{H}$, outputs the list of messages that P_i sends to every other party in the next

round. The compiled protocol is given in Figure 3.2, while the algorithms for cheater identification are in Figure 3.3.

In the compiled protocol (Figure 3.2), the parties additionally send a signature on the hash of each message. Signing the hash instead of the message itself allows our certificates for proving cheating to be succinct. If any signature is invalid, the receiving party broadcasts a complaint message, after which the sender must broadcast a valid message and signature to all parties, otherwise they are identified as a cheater. Note that if a malicious party tries to falsely complain about an honest sender, they merely force the sender to broadcast the correct message to all other parties; this does not cause a privacy issue, since the adversary had already received this message anyway.

In Figure 3.3, we give the algorithms for computing certificates and identify a corrupt party in case of cheating. The $\mathsf{Certify}$ algorithm takes as input the view of one party, and all parties' random tapes, and outputs a *partial certificate*, containing the signed hashes of any received messages that were inconsistent with the random tapes. Note that on its own, this does not prove cheating, because the inconsistency could be due to cheating in an earlier round from another party, which was not detected in this view. However, given all partial certificates, the $\mathsf{Identify}$ algorithm can then pinpoint a corrupt party by looking for the first inconsistency that was detected across all parties' views.

Figure 3.2: Protocol Π^{idc}

Let Π be the underlying passive protocol to be compiled, let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ be a collision-resistant hash function and $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ a signature scheme.

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 each round ρ of protocol Π
   (a) For every pair $i, j \in [n]$, $i \neq j$:
   - Let $m_{i,j}^{\rho}$ be the message which P_i should send to P_j according to NMF_{Π}
   - P_i sends $(m_{i,j}^{\rho}, \sigma_{i,j}^{\rho})$ to P_j, where $\sigma_{i,j}^{\rho} = \mathsf{Sig}_{\mathsf{sk}_i}(\rho \| P_i \| P_j \| \mathsf{H}(m_{i,j}))$
   - P_j checks that $\mathsf{Ver}(\mathsf{vk}_i, \sigma_{i,j}^{\rho}, m_{i,j}^{\rho}) = 1$. If this fails, or if P_i did not send a message, P_j broadcasts $(\mathsf{complain}, P_i)$. P_i then broadcasts $(m_{i,j}^{\rho}, \sigma_{i,j}^{\rho})$; if P_i fails to do this (or the signature is invalid) then all parties output (abort_i)
3. If the parties did not abort, they output the same as they would according to Π

Figure 3.3: Verification algorithms for Π^{idc}

$\mathsf{Certify}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{view}_i)$**:**

1. If view_i shows that P_i aborts, then output $\perp$.
2. Let R be the total number of rounds, and $\{m^{\rho}_{j,i}, \sigma^{\rho}_{j,i}\}_{j \neq i, \rho \in [R]}$ be the valid message/signature pairs contained in view_i.
3. Initialize sets $\mathcal{H}_1, \ldots, \mathcal{H}_n := \emptyset$.
4. For $\rho = 1, \ldots, R$:
   (a) For each $i' \in [n]$, compute the honest messages $(\widetilde{m}^{\rho}_{i',j})_{j \neq i'} = \mathsf{NMF}(i', \rho, r_{i'}, \mathcal{H}_{i'})$ and append $\widetilde{m}^{\rho}_{i',j}$ to the message history $\mathcal{H}_j$.
   (b) If $\mathsf{H}(m^{\rho}_{j,i}) \neq \mathsf{H}(\widetilde{m}^{\rho}_{j,i})$ for some $j \neq i$ then output $\mathsf{cert}_i := (\rho, P_j, P_i, \mathsf{H}(m^{\rho}_{j,i}), \sigma^{\rho}_{j,i})$.
5. If no certificate was obtained, output $\mathsf{cert}_i := \perp$.

$\mathsf{Identify}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{cert}_1, \ldots, \mathsf{cert}_n)$**:**

1. Using $r_1, \ldots, r_n$ and NMF, compute the set of honest messages $\{\widetilde{m}^{\rho}_{i,j}\}_{j \neq i}$ for $i \in [n], \rho \in [R]$ (as in $\mathsf{Certify}$).
2. Discard any certificate which is not of the form $\mathsf{cert}_i = (\rho_i, P_{j_i}, P_i, h_i, \sigma_i)$, for some $\rho_i \in [R], j_i \in [n], h_i \in \{0,1\}^{2\lambda}$ and signature σ_i, satisfying

$$\mathsf{Ver}(\mathsf{vk}_{j_i}, \sigma_i, (\rho_i \| P_{j_i} \| P_i \| h_i)) = 1 \quad \text{and} \quad h_i \neq \mathsf{H}(\widetilde{m}^{\rho_i}_{j_i,i})$$

3. If no certificates remain, output $\perp$.
4. Otherwise, let cert_i be the remaining certificate with the smallest ρ_i (to break ties, pick the smallest i). Output P_{j_i} as a cheater.

Theorem 1. *Suppose that Π securely computes $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ with passive security, and is well-defined with respect to malicious behaviour. Let H be randomly sampled from a family of collision-resistant hash functions, and $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be an EUF-CMA secure signature scheme. Then the compiled protocol Π^{idc} securely computes $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ with passive security, and has both (a) identifiable cheating from random tapes, and (b) consistent identifiability abort.*

The proof of Theorem 1 can be found in Appendix F.

3.3 Simplified Compiler Without Identifiable Abort

We can define a simple version of the compiler defined in Section 3.2 where the compiled protocol does not satisfies consistent identifiability abort but only the identifiable cheating from random tapes property. In this case for the compiled protocol there is no need to identify the misbehaving party. Therefore, if any party receives an invalid signature, instead of broadcasting a complaint, they simply output abort. We modify the compiled protocol as described above and we denote it with $\Pi^{\mathsf{w\text{-}idc}}$.

Theorem 2. *Suppose that Π securely computes $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ with passive security, and is well-defined with respect to malicious behaviour. Let H be randomly sampled from a family of collision-resistant hash functions, and $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be an EUF-CMA secure signature scheme. Then the compiled protocol $\Pi^{\mathsf{w\text{-}idc}}$ securely computes $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ with passive security, and has identifiable cheating from random tapes.*

3.4 Publicly Verifiable Cheating

In the setting of public verifiability, the parties need to able to produce a certificate that is verifiable by any third party, which is also *defamation-free*, meaning that no corrupt party can frame an honest party by maliciously generating a certificate. Note that identifiable cheating from random tapes does not necessarily enforce this, since in the security experiment, the honest parties' certificates cannot be tampered with by the adversary.[5]

Definition 8 (Publicly verifiable cheating from random tapes). *Let Π be a protocol between parties $P_1, \ldots, P_n$ in the PKI model, that takes no inputs. Suppose there are three deterministic polynomial-time algorithms:*

- $\mathsf{GatherEvidence}(\mathsf{view}_i)$*: On input a view view_i of some party P_i from an execution of Π, this outputs a partial certificate $\mathsf{cert}_i \in \{0,1\}^*$.*
- $\mathsf{Accuse}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{cert}_1, \ldots, \mathsf{cert}_n)$*: On input all parties' partial certificates and random tapes, this generates a global certificate $\mathsf{cert} \in \{0,1\}^*$.*
- $\mathsf{Sentence}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{cert})$*: On input a global certificate and all parties' public keys and random tapes, this either outputs the identity of a corrupted party, P_j, or an honest execution symbol $\perp$.*

Π supports publicly verifiable cheating *from random tapes if for any p.p.t. adversary $\mathcal{A}$ it holds that:*

$$\Pr[\mathtt{Exp}_{\mathcal{A},\Pi}^{\mathsf{pvc}}(\lambda) = 1] \leq \nu(\lambda)$$

where $\lambda \in \mathbb{N}$, ν is a negligible function and $\mathtt{Exp}_{\mathcal{A},\Pi}^{\mathsf{pvc}}(\lambda)$ is defined as follows:

Figure 3.4: Experiment $\mathtt{Exp}_{\mathcal{A},\Pi}^{\mathsf{pvc}}(\lambda, x)$

1. *$\mathcal{A}$ corrupts a set of parties $A \subset [n]$.*
2. *For each $i \in [n]$, sample a random tape r_i and a signing key pair $\mathsf{vk}_i, \mathsf{sk}_i$*
3. *The parties run Π, where party P_i is given r_i, sk_i and $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ as input. Let view_i denote the list of messages received by P_i.*
4. *If any honest party outputs abort during the execution, output 0. Otherwise, continue.*

[5] In fact, if an adversary can modify the honest parties' certificates then an honest party can be framed when using our compiler from the previous section.

5. *Generate the honest parties' partial certificates* $\mathsf{cert}_i = \mathsf{GatherEvidence}(\mathsf{view}_i)$*, for* $i \in [n] \setminus A$*, and send these and the honest random tapes to* $\mathcal{A}$*.*
6. $\mathcal{A}$ *outputs the partial certificates* cert_j*, for* $j \in A$*, and a global certificate* cert^**.*
7. *Generate an honest certificate* $\mathsf{cert} = \mathsf{Accuse}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{cert}_1, \ldots, \mathsf{cert}_n)$*.*
8. *Output* 1 *if one of the following holds:*
   (a) *(public verifiability) The execution of* Π *is dishonest with respect to* A*, and* $\mathsf{Sentence}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{cert}) = \perp$*; or*
   (b) *(defamation freeness)* $\mathsf{Sentence}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{cert}^*) \in \{P_i\}_{i \notin A}$

   Otherwise, output 0*.*

Compiler for Publicly Verifiable Cheating. To obtain publicly verifiable cheating, we use the same protocol transformation as for identifiable cheating without identifiable abort (Section 3.3), which simply adds a signature to the hash of every message. We then use the algorithms in Figure 3.5 to produce the publicly verifiable certificate of cheating. We will now briefly discuss them.

The algorithm $\mathsf{GatherEvidence}$ on input the view view_i of P_i outputs the set of hashed message/signature pairs contained in view_i.

The Accuse algorithm takes as inputs, public keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$, random tapes $r_1, \ldots, r_n$ and partial certificates $\mathsf{cert}_1, \ldots, \mathsf{cert}_n$. First of all Accuse validates the signature obtained in the certificate cert_i and sets $\mathsf{cert}_i = \perp$ if some signature is not valid. Note that the honest parties sends valid signature, otherwise cert_i would contain $\perp$. The Accuse algorithm using $r_1, \ldots, r_n$ and NMF will reconstruct the honest execution of $\Pi^{\mathsf{w\text{-}idc}}$ until some round ρ^* where the hash of the message in the honest execution does not corresponds to the hash h^* contained in cert_j, for $j \in [n]$. In this case Accuse produces a certificate which contains (1) the aggregate signatures (which is computed using the signature in the partial certificate) on the hash of the messages computed until round ρ^* (2) the hash and the signature of the malformed message (3) the indexs of the parties that sent/received that message and ρ^*. If there is no inconsistent message Accuse outputs $\perp$. We observe that Accuse algorithm is expecting as input honest generated random tapes $r_1, \ldots, r_n$. If an outer algorithm is invoking Accuse, then the outer algorithm has to ensure that this condition is satisfied.

Finally the $\mathsf{Sentence}$ algorithm (that can be run from any third part) takes as inputs a certificate $\mathsf{cert} = (\sigma_{\mathsf{Agg}}, \sigma^*, h^*, i^*, j, \rho^*)$, public keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ and honestly generated random tapes $r_1, \ldots, r_n$; moreover we are assuming that she is equipped with NMF. The $\mathsf{Sentence}$ algorithm using $r_1, \ldots, r_n$ and NMF will reconstruct the honest execution of $\Pi^{\mathsf{w\text{-}idc}}$ until the round ρ^* verifying that the corresponding aggregate signature σ_{Agg} is valid. Then $\mathsf{Sentence}$ verifies that σ^* is a valid signature on h^* and computes the honest message in round ρ^* that

parties P_{i^*} should have sent to party P_j if his hash does not match with h^* the $\mathsf{Sentence}$ algorithm outputs vk_{i^*} and $\perp$ otherwise.

Figure 3.5: Verification algorithms for $\Pi^{\mathsf{w\text{-}idc}}$

$\mathsf{GatherEvidence}(\mathsf{view}_i)$**:**

1. If view_i shows that P_i aborts, then output $\perp$.
2. Let R be the total number of rounds, and $\{\mathsf{H}(m^\rho_{j,i}), \sigma^\rho_{j,i}\}_{j\neq i,\rho\in[R]}$ be the set of hashed message/signature pairs contained in view_i.
3. Output the partial certificate $\mathsf{cert}_i = \{\mathsf{H}(m^\rho_{j,i}), \sigma^\rho_{j,i}\}_{j\neq i,\rho\in[R]}$.

$\mathsf{Accuse}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{cert}_1, \ldots, \mathsf{cert}_n)$**:**

1. Parse each cert_i as $\{\bar{h}^\rho_{j,i}, \bar{\sigma}^\rho_{j,i}\}_{j\neq i,\rho\in[R]}$.
2. If for some i, j, ρ there exists $\mathsf{Ver}(\mathsf{vk}_i, \bar{\sigma}^\rho_{i,j}, (\rho \| P_i \| P_j \| \bar{h}^\rho_{i,j})) \neq 1$, then set $\mathsf{cert}_i = \perp$.
3. Initialize ordered lists $\mathcal{G}, \mathcal{H}_1, \ldots, \mathcal{H}_n := \emptyset$.
4. For $\rho = 1, \ldots, R$:
   (a) For each $i \in [n]$, compute the honest messages $(\widetilde{m}^\rho_{i,j})_{j\neq i} = \mathsf{NMF}(i, \rho, r_i, \mathcal{H}_i)$ and append $\widetilde{m}^\rho_{i,j}$ to the message history $\mathcal{H}_j$.
   (b) If for all i, j (with $\mathsf{cert}_j \neq \perp$) it holds that $\mathsf{H}(\widetilde{m}^\rho_{i,j}) = \bar{h}^\rho_{i,j}$, then append each pair $(\bar{h}^\rho_{i,j}, \bar{\sigma}^\rho_{i,j})$ to the list of good signatures, $\mathcal{G}$. Otherwise, let $(\bar{h}^\rho_{i^*,j}, \bar{\sigma}^\rho_{i^*,j})$ be the smallest i^* for witch the above check fails. Output $\mathsf{cert} = (\mathsf{Agg}(\mathcal{G}), \bar{h}^\rho_{i^*,j}, \bar{\sigma}^\rho_{i^*,j}, i^*, j, \rho)$ and halt.
5. If no cheating was detected, output $\mathsf{cert} = \perp$.

$\mathsf{Sentence}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, r_1, \ldots, r_n, \mathsf{cert})$**:**

1. If $\mathsf{cert} = \perp$ then output $\perp$. Otherwise, parse $\mathsf{cert} = (\sigma_{\mathsf{Agg}}, h^*, \sigma^*, i^*, j, \rho^*)$.
2. Using $r_1, \ldots, r_n$ and NMF, compute the set of honest messages (as in Accuse) for rounds $\rho < \rho^*$, given by $\mathcal{M} = \{\mathsf{H}(\widetilde{m}^\rho_{i,j})\}_{i\in[n],j\neq i,\rho\in[\rho^*-1]}$.
3. Check that $\mathsf{AggVer}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, \sigma_{\mathsf{Agg}}, \mathcal{M}) = 1$. If this fails then output $\perp$.
4. Compute the honest message, $\widetilde{m}^{\rho^*}_{i^*,j}$, using the previously computed messages and r_{i^*}, and let $h = \mathsf{H}(\widetilde{m}^{\rho^*}_{i^*,j})$.
5. Check that $\mathsf{Ver}(\mathsf{vk}_{i^*}, \sigma^*, (\rho^* \| P_{i^*} \| P_j \| h^*)) = 1$ and $h \neq h^*$. If so, output vk_{i^*} as a cheater. Otherwise, output $\perp$.

Theorem 3. *Suppose that Π securely computes $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ with passive security, and is well-defined with respect to malicious behaviour. Let H be randomly sampled from a family of collision-resistant hash functions, and $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{AggVer})$ be a secure aggregate signature scheme. Then the compiled protocol $\Pi^{\mathsf{w\text{-}idc}}$ securely computes $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ with passive security and supports publicly verifiable cheating from random tapes (as described in Figure 3.5).*

We remark that if instead of using aggregate signatures, we use a standard signature scheme (and just concatenate all the signatures), our theorem still holds at a price of having less succinct certificates. The proof of Theorem 3 can be found in Appendix G.

4 Preprocessing with Identifiable Abort

In this section we are presenting a protocol $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$ which implements $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ 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 identifiability 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 $r_{i,j}$ 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 $j \neq c$. 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 4.1.

Figure 4.1: Protocol $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$

Let Π_{corr} be a protocol that computes $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ 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 $r_{i,j}$ in the j-th execution of the protocol. Let $R_{i,j}$ be the output of party P_i in execution j.
6. All parties send $(\mathsf{coinFlip}, 0)$ to $[\mathcal{F}_{\mathsf{FLIP}}]^{\mathsf{ida}}$ and obtain a uniformly random value $j^* \in [k]$.
7. For $j \in [k] \setminus \{j^*\}$ each party P_i
   (a) sends $(\mathsf{openAll}, (i,j))$ to $[\mathcal{F}_{\mathsf{FLIP}}]^{\mathsf{ida}}$.

(b) receives back $\boldsymbol{r}_j := (r_{1,j}, \ldots, r_{n,j})$ corresponding to the random tapes used in execution j from $[\mathcal{F}_{\mathsf{FLIP}}]^{\mathsf{ida}}$.
(c) computes $\mathsf{cert}_{i,j} \leftarrow \mathsf{Certify}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, \boldsymbol{r}_j, \mathsf{view}_{i,j})$, where $\mathsf{view}_{i,j}$ is P_i's view in the j-th execution of Π_{corr}.
(d) sends $\mathsf{cert}_{(i,j)}$ to $\mathcal{F}_{\mathsf{BC}}$.

8. Each party P_i receives certificates $(\mathsf{cert}_{1,j}, \ldots, \mathsf{cert}_{n,j})$ for each execution $j \neq j^*$ and computes $v_j \leftarrow \mathsf{Ver}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, \boldsymbol{r}_j, \mathsf{cert}_{1,j}, \ldots, \mathsf{cert}_{n,j})$. Let J be the set of indices with $v_j \neq \bot$. 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,j^*}.

Theorem 4. *Suppose protocol Π_{corr} securely implements $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ 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 and $[\mathcal{F}_{\mathsf{COM}}]^{\mathsf{ida}}$ be the ideal commitment functionality with identifiable abort. Let $\mathcal{F}_{\mathsf{BC}}$ be the broadcast functionality. Then $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$ implements $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ with security against covert adversaries, who corrupts $n-1$ parties, and deterrence factor $\epsilon = 1 - \frac{1}{k}$.*

The proof of Theorem 4 can be found in Appendix H.

5 Publicly Verifiable Preprocessing

In this section, we show how to compile any passively secure protocol Π_{corr} that has no private inputs into one that is secure against covert adversaries and that provides public verifiability. In contrast to covert security without public verifiability, here we do not require identifiable abort and are satisfied with unanimous abort, but in exchange we want to ensure that the honest parties obtain publicly verifiable certificates whenever a cheating attempt by the adversary is detected. The main challenge that needs to be overcome when constructing such protocols, is to obtain the certificates even if the adversary attempts to hide a failed cheating attempt by aborting the protocol execution.

As already mentioned in the technical overview in Section 1.2, we will employ time-lock puzzles for solving this problem. Recall that, for some parameter t, such puzzles allow a sender to encrypt a message in a manner that keeps it hidden for t rounds, but also allows the receiver to obtain the message after $t+1$ rounds without any further interaction with the sender.

On an intuitive level, we would like to follow the blueprint of our protocol for covert security without public verifiability, but instead of directly revealing which executions are being checked, we would like to reveal a time-lock puzzle that contains all the necessary information that is needed for checking $k-1$ random executions. Once the time-lock puzzle is revealed, all parties will sign the puzzle and broadcast the signatures to each other in the subsequent round. Since the adversary can not see the contents of the puzzle, it can either decide to abort *independently* of which executions are being checked or to sign the puzzle

and thereby potentially provide publicly verifiable evidence that can be used to incriminate its cheating attempt.

Puzzle generation. A naive approach for generating the puzzles would be the run an actively secure n-party protocol, which computes the desired puzzles inside the secure computation. Unfortunately this would incur a large practical overhead, since performing public-key operations, which are required for known instantiations of the puzzles, inside MPC protocols is expensive. Therefore, we would like to avoid the use of generic MPC as much as possible for the sake of concrete efficiency. Our protocol starts by letting all parties jointly generate $n \times k$ matrix C^{rnd} of commitments $\mathsf{com}_{i,j}$ to random tapes $r_{i,j}$ for $i \in [n]$ and $j \in [k]$, where party P_i knows the corresponding decommitments for all $r_{i,j}$ with $j \in [k]$. After all parties have executed k instances of Π_{corr} with the appropriate random tapes, each party P_i (with view $\mathsf{view}_{i,j}$ of the j-th execution) for each of the k executions runs $\mathsf{cert}_{i,j} \leftarrow \mathsf{GatherEvidence}(\mathsf{view}_{i,j})$ to generate a partial certificate. Then, the parties jointly run an actively secure protocol Π_{check}, where each party inputs all of its decommitments from C^{rnd} and all partial certificates. The protocol produces an authenticated secret sharing $(s_1, \ldots, s_n)$ of all decommitments and partial certificates belonging to $k-1$ random executions, where party P_i obtains s_i. Each party commits to its share s_i and broadcasts the commitment $\mathsf{com}_i^{\mathsf{share}}$ to everybody. Finally, each party P_i broadcasts a time-lock puzzle puz_i containing the decommitment $\mathsf{dec}_i^{\mathsf{share}}$ of $\mathsf{com}_i^{\mathsf{share}}$ and all parties sign the puzzles.

In terms of efficiency, it is important to note that the computation of Π_{check} only depends on the number of parties and only performs fast symmetric-key operations inside of the protocol.

In terms of security, the intuition is as follows: A cheating adversary, can provide one or more incorrect decommitments for C^{rnd} as its input to Π_{check}. If the adversary guesses incorrectly which execution is not being checked, then it will eventually have to sign a time-lock puzzle that contains an incriminating decommitment for some commitment in C^{rnd}. Alternatively, the adversary can provide honest inputs to Π_{check}, but then decide to produce a commitment $\mathsf{com}_i^{\mathsf{share}}$, which contains a modified secret share. In this case, the protocol will *always* abort, since the reconstruction of the checked executions will always fail and thus we do not need to produce a publicly verifiable certificate. If the adversary produces the commitment $\mathsf{com}_i^{\mathsf{share}}$ correctly, but puts the incorrect decommitment information into the corresponding puzzle, then it will again incriminate itself by signing the puzzle.

The remainder of the protocol. Once the puzzles and the corresponding signatures have been sent around, the parties broadcast the decommitments $\mathsf{dec}_i^{\mathsf{share}}$ that are stored inside the puzzles. Given the shares of all parties, each individual party can check $k-1$ of the executions. If, and only if, some party stops responding at this stage of the protocol, then the honest parties need to solve one or more puzzles and either obtain the necessary information that was not sent from them or produce a certificate, which shows that some malicious party tampered with its puzzle in a malicious manner. Equipped with this high-

level intuition we are not ready to present our protocol which is described in Figure 5.1 and the corresponding Judge algorithm is described in Figure 5.2.

Figure 5.1: Protocol $[\Pi_{\mathsf{corr}}]^{\mathsf{pvcov}}$

1. Each party P_i receives a signing key sk_i and verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ from the PKI. Let $\overrightarrow{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$.

Committed Coin Flip:

2. All parties jointly and repeatedly execute the committed coin flip protocol Π_{flip}, such that each party P_i receives an output

$$C^{\mathsf{rnd}} = \begin{pmatrix} \mathsf{com}_{1,1}, \ldots, \mathsf{com}_{1,k} \\ \vdots \\ \mathsf{com}_{n,1}, \ldots, \mathsf{com}_{n,k} \end{pmatrix} \text{ and } D_i = \left(\mathsf{dec}_{i,1}, \ldots, \mathsf{dec}_{i,k}\right),$$

   where $\mathsf{com}_{i,j}$ is a commitment to $r_{i,j}$ with decommitment $\mathsf{dec}_{i,j}$.

Protocol Execution:

3. All parties jointly execute Π_{corr} in parallel k times, where party P_i uses random tape $r_{i,j}$ in the j-th execution of the protocol. Let $R_{i,j}$ be the output of party P_i in execution j. Let $\mathsf{view}_{i,j}$ be the view of P_i in the j-th execution of Π_{corr}. After the executions, for $j \in [k]$, each P_i runs $\mathsf{cert}_{i,j} \leftarrow \mathsf{GatherEvidence}(\mathsf{view}_{i,j})$.

Puzzle Generation:

4. Each party P_i picks a uniformly random encryption keys $K_{i,1}, \ldots, K_{i,k}$ and broadcasts encryptions $c_{i,j} \leftarrow \mathsf{sEnc}_{K_{i,j}}(\mathsf{dec}_{i,j}, \mathsf{cert}_{i,j})$ for $j \in [k]$. Let CT be the set of all $n \times k$ ciphertexts.
5. All parties jointly execute Π_{check} where party P_i uses $(K_{i,1}, \ldots, K_{i,k})$ as input. Each party P_i obtains output s_i, which is the i-th share of

$$\boldsymbol{s} = \left(\begin{pmatrix} K_{1,\gamma(1)}, \ldots, K_{n,\gamma(1)} \\ \vdots \\ K_{1,\gamma(k-1)}, \ldots, K_{n,\gamma(k-1)} \end{pmatrix}, \gamma \right)$$

   where γ is a uniformly random permutation on $[k]$.
6. Each P_i computes $\mathsf{dec}_i^{\mathsf{share}}$ and $\mathsf{com}_i^{\mathsf{share}}$ by using Π_{com} to commit to s_i and broadcasts $\mathsf{com}_i^{\mathsf{share}}$. Let $C^{\mathsf{share}} = \left(\mathsf{com}_1^{\mathsf{share}}, \ldots, \mathsf{com}_n^{\mathsf{share}}\right)$.
7. Each P_i generates a puzzle $\mathsf{puz}_i \leftarrow \mathsf{pGen}\left(t, \left(\mathsf{dec}_i^{\mathsf{share}}\right)\right)$, where $t = 5$, commits to puz_i using Π_{com} obtaining $\mathsf{com}_i^{\mathsf{puz}}, \mathsf{dec}_i^{\mathsf{puz}}$, and broadcasts $\mathsf{com}_i^{\mathsf{puz}}$. Let $\overrightarrow{\mathsf{puz}}$ be the vector of all puzzles $\mathsf{puz}_1, \ldots, \mathsf{puz}_n$.
8. After receiving $\mathsf{com}_j^{\mathsf{puz}}$ for all $j \neq i$, each P_i broadcasts puz_i and $\mathsf{dec}_i^{\mathsf{puz}}$. Each party checks whether all the puzzle commitments it receives are valid decommitments and aborts if this is not the case.

9. Each party P_i computes signature $\sigma_i \leftarrow \mathsf{Sig}_{\mathsf{sk}_i}\left(C^{\mathsf{rnd}}, C^{\mathsf{share}}, CT, \overrightarrow{\mathsf{puz}}\right)$ and broadcasts it to every other party. If any party obtains an invalid signature or obtains a signature in the incorrect round, then the party aborts.

Checking Phase:

10. Each P_i broadcasts $\mathsf{dec}_i^{\mathsf{share}}$. Let $J \subset [n]$ be the set indices belonging to parties that did not broadcast this message. If $J \neq \emptyset$, then each party P_i solves all puzzles puz_j with $j \in J$. If one or more puzzles are not solvable in time t, then P_i outputs certificate $\mathsf{cert} = (1, j^*, \sigma_{j^*}, C^{\mathsf{rnd}}, C^{\mathsf{share}}, CT, \overrightarrow{\mathsf{puz}})$, where j^* is the smallest index belonging to a not solvable puzzle, and aborts. If all relevant puzzles solvable, then replace missing decommitments with the ones from the solved puzzles.
11. Let S be the set of invalid decommitments for C^{share}. Solve corresponding puzzles to see whether they contain the valid decommitment. If yes, then continue as below using the valid decommitment, otherwise output $\mathsf{cert} = (2, j^*, \sigma_{j^*}, C^{\mathsf{rnd}}, C^{\mathsf{share}}, CT, \overrightarrow{\mathsf{puz}})$, where j^* is the smallest index belonging to a not valid decommitment, and terminate.
12. If any of the decommitted shares is invalid, then abort.
13. Each party uses the shares to reconstruct $K_{1,\gamma(j)}, \ldots, K_{n,\gamma(j)}$ for $j \in [k-1]$.
14. Each party P_i
    - checks whether any of the decryptions fail and if it does then we set the corresponding plaintext to be $\perp$.
    - checks whether the reconstructed random tapes and decommitments for each execution $j \in [k-1]$ match the corresponding commitments in matrix C^{rnd}. If not, let j^* be the smallest execution index with a mismatch and let i^* be the smallest index of a party within that execution that produces a mismatch. Every party outputs certificate $\mathsf{cert} = (3, i^*, \sigma_{i^*}, C^{\mathsf{rnd}}, C^{\mathsf{share}}, CT, \overrightarrow{\mathsf{puz}})$.
    - Let $\overrightarrow{\mathsf{cert}}_{\gamma(j)}$ be the vector of partial certificate belonging to execution $\gamma(j)$. Let $\overrightarrow{r}_{\gamma(j)} = (r_{1,\gamma(j)}, \ldots, r_{n,\gamma(j)})$ be the vector of random tapes belonging to execution $\gamma(j)$.
      Each P_i, for each $j \in [k-1]$ computes

      $$\mathsf{cert}'_{\gamma(j)} \leftarrow \mathsf{Accuse}(\overrightarrow{\mathsf{pk}}, \overrightarrow{r}_{\gamma(j)}, \overrightarrow{\mathsf{cert}}_{\gamma(j)}).$$

      Let J be the set of indices j with $\mathsf{Sentence}(\overrightarrow{\mathsf{pk}}, \overrightarrow{r}_{\gamma(j)}, \mathsf{cert}'_{\gamma(j)}) \neq \perp$. If $J \neq \emptyset$, then let j^* be the smallest index, and let

      $$i^* \leftarrow \mathsf{Sentence}(\overrightarrow{\mathsf{pk}}, \overrightarrow{r}_{\gamma(j^*)}, \mathsf{cert}'_{\gamma(j^*)})$$

      and output $\mathsf{cert} =$

      $$\begin{pmatrix} 4, i^*, \sigma_{i^*}, C^{\mathsf{rnd}}, C^{\mathsf{share}}, CT, \overrightarrow{\mathsf{puz}}, \\ \{\mathsf{dec}_{i,\gamma(j^*))}\}_{i\in[n]}, \mathsf{cert}'_{\gamma(j^*)}, j^* \end{pmatrix}$$

Output Phase:

15. If no misbehavior was detected, then each party P_i outputs $R_{i,\gamma(k)}$.

Figure 5.2: Judge has input $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ and cert

1. The judge parses cert as (c, aux)
   - If $c \in \{1,2,3\}$, then parse aux as $(i^*, \sigma_{i^*}, C^{\mathsf{rnd}}, C^{\mathsf{share}}, CT, \overrightarrow{\mathsf{puz}})$.
   - If $c = 4$, then parse aux as
   $$\left(i^*, \sigma_{i^*}, C^{\mathsf{rnd}}, C^{\mathsf{share}}, CT, \overrightarrow{\mathsf{puz}}, \{\mathsf{dec}_{i,\gamma(j^*)}\}_{i\in[n]}, \mathsf{cert}'_{\gamma(j^*)}, j^*\right).$$
   - If certificate not well-formed, then judge outputs $\perp$.
   - The judge checks whether $\mathsf{Ver}\big(\mathsf{pk}_{i^*}, \sigma_{i^*}, (C^{\mathsf{rnd}}, C^{\mathsf{share}}, CT, \overrightarrow{\mathsf{puz}})\big) = 1$ and otherwise output $\perp$.
2. Depending on the error index c do the following:

   $\underline{\mathbf{c = 1}}$:
   - The judge attempts to solve $\mathsf{puz}_{i^*} \in \overrightarrow{\mathsf{puz}}$ in time $t = 5$. If the puzzle is not solvable within the given time, output pk_{i^*} and otherwise output $\perp$.

   $\underline{\mathbf{c = 2}}$:
   - Judge solves puz_{i^*} to obtain decommitments corresponding to commitments of P_{i^*} in C^{share}. If any of the decommitments is invalid, then the judge outputs pk_{i^*}.

   $\mathbf{c} \in \{\mathbf{3}, \mathbf{4}\}$:
   - The judge solves puzzles $\mathsf{puz}_1, \ldots, \mathsf{puz}_n$ to obtain decommitments corresponding to commitments in C^{share}. If any decommitment invalid, then output $\perp$.
   - The judge obtains the shares of $k-1$ keys. If any share is invalid, then output $\perp$. Otherwise the judge reconstructs the permutation γ and the keys $K_{i,\gamma(1)}, \ldots, K_{i,\gamma(k-1)}$ for each $i \in [n]$.
   - It decrypts the ciphertexts in CT corresponding to the available keys. For all decryptions that fail, we set the corresponding plaintext to be $\perp$. If there exists a j such that the ciphertext $c_{i^*,j}$ contains a decommitment that does not match the corresponding committed random tape in C^{rnd}, then output pk_{i^*}.

     $\underline{\mathbf{c = 3}}$: If nothing bad happened till now, then the judge outputs $\perp$.

     $\underline{\mathbf{c = 4}}$: Let $\overrightarrow{r}_{\gamma(j^*)}$ be the vector of random tapes belonging to execution j^*. Output $\mathsf{Sentence}(\overrightarrow{\mathsf{pk}}, \overrightarrow{r}_{\gamma(j^*)}, \mathsf{cert}'_{\gamma(j^*)})$.

Theorem 5. *Suppose protocol Π_{corr} securely implements $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ with passive security, has public verifiability cheating from random tapes. Let Π_{com}, Π_{check} and Π_{flip} be protocols that securely implement $\mathcal{F}_{\mathsf{COM}}$, $\mathcal{F}_{\mathsf{CHECK}}$ and $\mathcal{F}_{\mathsf{FLIP}}$ respectively with UC-security. Let $(\mathsf{pGen}, \mathsf{pSol})$ be a B-secure time-lock puzzle for*

$B = 5$. *Let* $(\mathsf{sEnc}, \mathsf{sDec})$ *be a CPA-secure symmetric encryption scheme and let* $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *be an EUF-CMA secure signature scheme. Furthermore, assume all parties have access to a broadcast channel. Then* $[\Pi_{\mathsf{corr}}]^{\mathsf{pvcov}}$ *implements* $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ *with covert security and public verifiability against an adversary, who corrupts* $n-1$ *parties, and deterrence factor* $\epsilon = 1 - \frac{1}{k}$.

The ideal functionalities used in the theorem statement can be found in Section C.4 and the proof of the theorem can be found in Appendix I.

6 Instantiating the Preprocessing and Online Phases

We now show how to instantiate the passively secure preprocessing phase we need, as well as an online phase, filling in the missing pieces for our constructions with covert security. We focus on instantiating using the BMR protocol for constant-round MPC of binary circuits, achieving covert security with either identifiable abort or public verifiability. However, we also discuss other use-cases, such as variants of the SPDZ protocol, and efficient preprocessing of one-time truth tables.

Combining the Online and Preprocessing Phases. In our instantiations, we will take a covertly secure preprocessing protocol output by one of our compilers, and combine it with an actively secure online phase. In Appendix D.4, we show that this general approach of combining two such protocols is sound, that is, it leads to a protocol with covert security overall. Furthermore, if the original protocols also have identifiable abort, then so does the combined protocol.

Instantiating with BMR and Public Verifiability. In the BMR paradigm, the parties jointly construct a secret-sharing of a garbled circuit in the preprocessing phase, such that no single party knows all of the secret randomness. Then, in the online phase, the garbled circuit is reconstructed, and all parties locally evaluate it. Note that the garbled circuit shares are not authenticated, so corrupt parties may send incorrect shares in the online phase; nevertheless, as long as the preprocessing was done correctly, this still gives an *actively secure* online phase [23] with abort.

By defining the preprocessing to only output shares of the garbled circuit, rather than the garbled circuit itself, we save efficiency with our covert security compiler. The parties only ever compute a single garbled circuit, regardless of the repetition factor k, which reduces computation and communication. The passively secure protocol which we compile is a variant of the semi-honest preprocessing from [8] (incorporating some later optimizations [23]). For completeness, we describe the full protocol and functionality in Appendix D, and also show that the functionality can be naturally described to fit the "reverse-sampleable" requirement.

In Appendix D, we also describe an optimization to the BMR preprocessing, which reduces the number of oblivious transfers by 25% compared with previous passively secure works [8, 12]. The high-level idea is to garble the circuit from

the input layer, in a top-down manner, instead of garbling each AND gate independently. By exploiting information from the previous gates, we show that this allows one set of OTs per AND gate to be removed.

Instantiating with BMR and Identifiable Abort. To achieve identifiable abort with covert preprocessing, we need an online phase that is secure in this model, with active security. A previous BMR-based protocol satisfies active security and identifiable abort [6], however, this requires a more complex preprocessing phase involving homomorphic commitments. We show that this can be avoided for the case of covert security, which greatly simplifies the protocol.

We propose to modify the previous preprocessing so that each party is *committed to* its garbled circuit share, as well as its keys for the input wires, with commitments given to all other parties. This ensures that in the online phase, there is no way to cheat when opening the garbled circuit. Since the preprocessing protocol itself only needs to be passively secure, it is very easy to achieve this functionality by just having the parties broadcasts their commitments.

One technical challenge of this approach, however, is that we need the preprocessing functionality to be reverse-sampleable. To satisfy this, the functionality would have to allow a corrupt party to *choose* the commitments it receives from honest parties, and then the functionality would sample the honest parties' outputs (i.e. decommitment information) consistently with these commitments. Unfortunately, this strong form of equivocation is not possible with standard commitments. Instead, we rely on *unanimously identifiable commitments* [26], which can be built information-theoretically in such a way that a commitment and message can be sampled *before* the corresponding decommitment. A slight downside is that we now need an interactive protocol to generate the commitments, however, we only need to perform a small number of commitments, and this overhead is independent of the circuit size. See Appendix D.3 for details.

Other Instantiations: SPDZ with Identifiable Abort and TinyTable. Since our compilers are general, they can be applied to any number of preprocessing-based MPC protocols. Here, we mention just a couple of examples where we expect to see a large improvement compared with active security.

SPDZ [18, 15] is a non-constant-round protocol for evaluating arithmetic circuits, with an expensive preprocessing phase based on homomorphic encryption. In [15], a covertly secure preprocessing was given, however, it neither achieved public verifiability nor identifiable abort. We can easily fix this by applying our compilers to a passively secure version of the SPDZ preprocessing. For covert security with public verifiability (without identifiable abort), we can use the standard online phase of SPDZ, combined with a simplified passive preprocessing protocol which we compile. If we want covert security with identifiable abort, then we instead use a variant of the online phase with identifiable abort [5], and can also simplify the preprocessing from that protocol to have passive security. Since the preprocessing is by far the bottleneck (and much more expensive than in regular SPDZ), this should be a more practical approach to achieving identifiable abort in SPDZ than protocols with full active security [5, 13, 31].

Another example is preprocessed, authenticated one-time truth tables, as used in the TinyTable protocol [16]. These allow an online phase with very efficient "table lookup" operations, where the table is public but the index is secret, and can for example be applied to AES S-boxes. Unfortunately, generating the secret tables with active security is very expensive. Boyle et al. [11] proposed to use distributed point functions for improved efficiency, since these allow the communication and storage costs to be *logarithmic* in the table size, instead of linear. However, no efficient, actively secure protocols for setting up distributed point functions are known. Instead, applying our covert compiler can lead to a covertly secure preprocessing for TinyTable, with much smaller costs than active security. We leave a detailed analysis of this to future work.

7 Efficiency Analysis

We now take a look at the concrete efficiency of our compilers for covert security, when applied to our passively secure BMR protocol from the previous section.

Metrics. We consider $n = 5$ parties, securely computing a Boolean circuit with 100000 AND gates, 128 bits of input per party and 128 output bits. We measure the bandwidth costs, per party, and also count the total number of OTs that are needed, since for the preprocessing phase in BMR, this gives a rough idea of the main computational costs. When measuring the communication cost of a (correlated) OT, we consider two different methods: in *regular OT* (R-OT), we use standard OT extension techniques [25, 1] to create the OT on random strings, which costs $\approx \lambda = 128$ bits of communication. In *silent OT* (S-OT), we use the recent, silent OT extension method based on a variant of the LPN assumption [11]; this can cost as little as 0.1 bits (on average) per random OT or $\mathcal{F}_{\mathsf{cot}}$ [10, 34], but has a higher computational cost.

In both cases, we ignore the cost of the setup phase for generating a small number of seed OTs. Our covert protocols have some additional costs such as coin-tossing and adding a signature to every message. Note that coin-tossing can easily be implemented with hash functions in the random oracle model, and since our protocols are constant-round, the number of signatures is very small, so for large circuits these costs will be insignificant.

Passive Security and Covert Security Without Identifiability (Table 1). As a baseline, we take the passive secure BMR protocol by [8], with an optimization to the way the OTs are generated (as described in [12]). We can see that our optimized passive protocol reduces bandwidth in the circuit-independent preprocessing by around 25%. When also factoring in sending the garbled circuit, for the regular-OT case this translates to a total saving of around 10%.

With regular OT extension, our covertly secure protocol with $k = 3$ repetitions (deterrrence factor $\frac{2}{3}$) has around 3x the preprocessing cost of the passive protocol, but the online cost remains the same. For the overall cost, the overhead is around 40%. Meanwhile, state-of-the-art actively secure protocols based on [23, 32, 33] have a total bandwidth around twice that of the covert protocol, and need 3x as many OTs. When using silent OT (S-OT) based on LPN, since

Protocol	Preprocessing			Online	Total	
	# OTs	R-OT	S-OT		R-OT	S-OT
Passive [8]	8.00	128.80	0.90	256.21	385.01	257.11
Passive (ours)	6.01	96.81	0.68	256.21	353.01	256.88
Covert (non-id)	18.00	290.42	2.03	256.21	546.62	258.23
Active	54.00	576.30	9.05	256.21	832.51	265.26

Table 1. Bandwidth costs (in MB) and OT costs (millions of OTs) for passive, covert and actively secure protocols without identifiable abort, in a Boolean circuit with 100 thousand ANDs, with covert deterrence factor $\frac{2}{3}$

OTs are so cheap, the bandwidth costs of adding active security are dwarfed by just the cost of sending the garbled circuit. However, the actively secure protocols still require a large number of OTs, which will add to the local computation costs.

In Table 2 in Appendix E, we also compare costs as the number of repetitions k is increased. Depending on the setting, our covert protocol continues to have smaller bandwidth and OT costs than an actively secure protocol up to around $k = 8$, while beyond that it is seems preferable to go for active security.

Public Verifiability. Our construction with public verifiability has similar costs to our covertly secure protocol without identifiable abort, the main differences being (1) The parties run a small, actively secure protocol Π_{check} to select which execution to open, and (2) Each party needs to construct a time-lock puzzle. The protocol Π_{check} can be done with around $nk\lambda$ AND gates, so when evaluating large circuits we do not expect this to be a bottleneck. Generating a single time-lock puzzle is also relatively cheap, and we note that the parties only have to work to solve the puzzles when there is a dishonest party. Note that when using aggregate signatures, the size of a certificate that is given to the judge is constant.

Identifiable Abort. For covert security with identifiable abort, the main difference in our compiled protocol is that the parties need to generate UIC commitments of their garbled circuit shares in the preprocessing. As argued in Appendix D, this can be done with passive security relatively efficiently, and we note that the number (and size) of commitments is independent of the circuit size. Compared with an actively secure protocol with identifiable abort [6], our covert protocol is much simpler, since we avoid the need to generate and broadcast homomorphic commitments to every wire key in the circuit, as well as additional commitments and checks in the preprocessing phase (on top of the standard active secure protocol) from [6]. Since broadcast is the dominating factor, and our preprocessing stage does not need any broadcasts (except in a dishonest execution) we expect our protocol to be highly competitive when identifiable abort is desired.

Finally, we can also compare our protocol with that of Goyal et al [22], if it was fixed to prevent the bug we pointed out. Since fixing this requires every

message of the underlying GMW protocol that is compiled to be broadcast, while our compiler avoids all broadcasts, we clearly improve efficiency by at least a factor n, for n parties.

Acknowledgements

This work has been supported by a DFF Sapere Aude Grant 9064-00068B, the Concordium Blockhain Research Center, Aarhus University, and a starting grant from Aarhus University Research Foundation.

References

1. G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS 2013: 20th Conference on Computer and Communications Security*. ACM Press, Nov. 2013.
2. G. Asharov and C. Orlandi. Calling out cheaters: Covert security with public verifiability. In *Advances in Cryptology – ASIACRYPT 2012*, Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2012.
3. Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC 2007: 4th Theory of Cryptography Conference*, Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2007.
4. C. Baum, B. David, R. Dowsley, J. B. Nielsen, and S. Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In *EUROCRYPT 2021*, 2021. https://eprint.iacr.org/2020/537.
5. C. Baum, E. Orsini, and P. Scholl. Efficient secure multiparty computation with identifiable abort. In *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, Lecture Notes in Computer Science. Springer, Heidelberg, Oct. / Nov. 2016.
6. C. Baum, E. Orsini, P. Scholl, and E. Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In *Advances in Cryptology – CRYPTO 2020, Part II*, Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2020.
7. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*. ACM Press, May 1990.
8. A. Ben-Efraim, Y. Lindell, and E. Omri. Optimizing semi-honest secure multiparty computation for the internet. In *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. ACM Press, Oct. 2016.
9. N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters. Time-lock puzzles from randomized encodings. In *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*. Association for Computing Machinery, Jan. 2016.
10. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019: 26th Conference on Computer and Communications Security*. ACM Press, Nov. 2019.
11. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *Advances in Cryptology – CRYPTO 2019, Part III*, Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2019.

12. L. Braun, D. Demmler, T. Schneider, and O. Tkachenko. Motion - a framework for mixed-protocol multi-party computation. Cryptology ePrint Archive, Report 2020/1137, 2020. https://eprint.iacr.org/2020/1137.
13. R. K. Cunningham, B. Fuller, and S. Yakoubov. Catching MPC cheaters: Identification and openability. In *ICITS 17: 10th International Conference on Information Theoretic Security*, Lecture Notes in Computer Science. Springer, Heidelberg, Nov. / Dec. 2017.
14. I. Damgård, M. Geisler, and J. B. Nielsen. From passive to covert security at low cost. In *TCC 2010: 7th Theory of Cryptography Conference*, Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2010.
15. I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS 2013: 18th European Symposium on Research in Computer Security*, Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2013.
16. I. Damgård, J. B. Nielsen, M. Nielsen, and S. Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *Advances in Cryptology – CRYPTO 2017, Part I*, Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2017.
17. I. Damgård, C. Orlandi, and M. Simkin. Black-box transformations from passive to covert security with public verifiability. In *Advances in Cryptology – CRYPTO 2020, Part II*, Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2020.
18. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2012.
19. N. Ephraim, C. Freitag, I. Komargodski, and R. Pass. Non-malleable time-lock puzzles and applications. Cryptology ePrint Archive, Report 2020/779, 2020. https://eprint.iacr.org/2020/779.
20. S. Faust, C. Hazay, D. Kretzler, and B. Schlosser. Generic compiler for publicly verifiable covert multi-party computation. In *EUROCRYPT 2021*, 2021. https://eprint.iacr.org/2021/251.
21. O. Goldreich and A. Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, (3), June 1996.
22. V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *Advances in Cryptology – EUROCRYPT 2008*, Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2008.
23. C. Hazay, P. Scholl, and E. Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *Advances in Cryptology – ASIACRYPT 2017, Part I*, Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2017.
24. C. Hong, J. Katz, V. Kolesnikov, W. Lu, and X. Wang. Covert security with public verifiability: Faster, leaner, and simpler. In *Advances in Cryptology – EUROCRYPT 2019, Part III*, Lecture Notes in Computer Science. Springer, Heidelberg, May 2019.
25. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO 2003*, Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2003.
26. Y. Ishai, R. Ostrovsky, and H. Seyalioglu. Identifying cheaters without an honest majority. In *TCC 2012: 9th Theory of Cryptography Conference*, Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2012.

27. Y. Ishai, R. Ostrovsky, and V. Zikas. Secure multi-party computation with identifiable abort. In *Advances in Cryptology – CRYPTO 2014, Part II*, Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2014.
28. V. Kolesnikov and A. J. Malozemoff. Public verifiability in the covert model (almost) for free. In *Advances in Cryptology – ASIACRYPT 2015, Part II*, Lecture Notes in Computer Science. Springer, Heidelberg, Nov. / Dec. 2015.
29. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. 1996.
30. H. A.-J. Seyalioglu. *Reducing trust when trust is essential*. PhD thesis, UCLA, 2012.
31. G. Spini and S. Fehr. Cheater detection in SPDZ multiparty computation. In *ICITS 16: 9th International Conference on Information Theoretic Security*, Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2016.
32. X. Wang, S. Ranellucci, and J. Katz. Global-scale secure multiparty computation. In *ACM CCS 2017: 24th Conference on Computer and Communications Security*. ACM Press, Oct. / Nov. 2017.
33. K. Yang, X. Wang, and J. Zhang. More efficient MPC from improved triple generation and authenticated garbling. In *ACM CCS 20: 27th Conference on Computer and Communications Security*. ACM Press, 2020.
34. K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang. Ferret: Fast extension for correlated ot with small communication. Cryptology ePrint Archive, Report 2020/924, 2020. https://eprint.iacr.org/2020/924.

A Impossibility Results for Security Against Covert Adversaries

We show that there exists an n-party functionality which can not be correctly computed by parties, who have oracle access to arbitrary two-party computations and broadcast, in the presence of a covert adversary, who corrupts two thirds of the parties. The proof strategy is essentially identical to a previous proof by Ishai, Ostrovsky, and Seyalioglu [26][6], which shows the same impossibility holds if one aims for active security with identifiable abort. For the sake of completeness, we provide a full write-up of our proof here. The second half of our proof is a little bit more concise and arguably simpler than the original proof, but we do not claim any particular novelty in terms of ideas when compared to the previous impossibility result.

A.1 Probability Theory Basics

Here we just recall some basics about probability theory, which will be used in the following impossibility result.

Lemma 6 (Chain Rule). *Let $A, B \subseteq \Omega$, where Ω is a sample space, then*

$$\Pr[A, B] = \Pr[B \mid A] \cdot \Pr[A].$$

Lemma 7 (Bayes' Theorem). *Let $A, B \subseteq \Omega$, where Ω is a sample space, then*

$$\Pr[A \mid B] = \frac{\Pr[A, B]}{\Pr[B]}.$$

Corollary 8. *Let $A, B, C \subseteq \Omega$, where Ω is a sample space, then*

$$\Pr[A \mid B, C] = \frac{\Pr[A, B \mid C]}{\Pr[B \mid C]}.$$

Proof.

$$\Pr[A \mid B, C] = \frac{\Pr[A, B, C]}{\Pr[B, C]} = \frac{\Pr[A, B \mid C] \cdot \Pr[C]}{\Pr[B \mid C] \cdot \Pr[C]} = \frac{\Pr[A, B \mid C]}{\Pr[B \mid C]}.$$

□

Lemma 9 (Law of Total Probability). *Let $B_1, \ldots, B_n$ be a partitioning of the sample space Ω and let $A \subseteq \Omega$, then*

$$\Pr[A] = \sum_{i=1}^{n} \Pr[A, B] = \sum_{i=1}^{n} \Pr[A \mid B] \cdot \Pr[B].$$

[6] The actual proof of their impossibility result can be found in Seyalioglu's thesis [30].

A.2 Impossibility Result

Theorem 10. *Let $\mathcal{F}_{\mathsf{2PC}}$ be an arbitrary two-party ideal functionality and let $\mathcal{F}_{\mathsf{BC}}$ be the ideal broadcast functionality. There exists a three-party functionality $\mathcal{F}$, which cannot be implemented with unconditional security against a covert adversary, who corrupts two of the parties, in the $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{BC}})$-hybrid model.*

Proof. Let $\langle \cdot, \cdot \rangle$ denote the inner product over $\mathbb{F}_2$. Consider the following three-party functionality

$$\mathcal{F}\begin{pmatrix} \boldsymbol{b}_0, \boldsymbol{b}_1 \\ \boldsymbol{c} \\ d \end{pmatrix} = \begin{pmatrix} \perp \\ \langle \boldsymbol{b}_0, \boldsymbol{c} \rangle, \langle \boldsymbol{b}_1, \boldsymbol{c} \rangle \\ \boldsymbol{b}_d \end{pmatrix},$$

where $\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{c} \in \{0,1\}^2$ and $d \in \{0,1\}$.

Assume towards contradiction that there exists a protocol Π that implements $\mathcal{F}$ with unconditional security against a covert adversary, who can corrupt up to two parties, in the $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{BC}})$-hybrid model. Let us start by considering two different possible adversaries $\mathcal{A}_1$ and $\mathcal{A}_3$, who corrupt P_1 or P_3 respectively. Adversary $\mathcal{A}_i$ for $i \in \{1,3\}$ behaves honestly and follows the protocol description towards P_2, but fully ignores P_j for $j \in \{1,3\} \setminus \{i\}$. A bit more concretely, in every round $\mathcal{A}_i$ calls the next-message functionality honestly using his current view, the latest message it receives from P_2, and instead of P_j's message it always inputs $\perp$, which we interpret as the "no message received special symbol". It sends the computed message to P_2, but drops whatever it is supposed to send to P_j. If P_j accuses P_i of not sending any messages, then P_i does the same and accuses P_j.

We observe that the view of P_2 is distributed *identically* in both an execution of Π in the presence of $\mathcal{A}_1$ and an execution of Π in the presence of $\mathcal{A}_3$, since it has no way of telling who of the two parties is lying. In the following we will call this modified execution Π'.

Claim 11. *For any set of inputs (x_1, x_2, x_3) to Π' and any $i, j \in [3]$, it holds that*

$$\Pr[P_i \text{ outputs } \{(\mathsf{corrupted}, j), (\mathsf{abort}, j)\}] \leq \mathsf{negl}(\delta),$$

where the probability is taken over the random coins of the parties and δ is the statistical security parameter.

Proof. We first observe that either none of the honest parties abort (or send the corrupted command) or all of them abort by outputting the same index. Since P_2 does not know who to accuse, it can not abort the protocol execution. □

Claim 12. *For any set of inputs (x_1, x_2, x_3), let (y_1, y_2, y_3) be the output of $\mathcal{F}$ and (z_1, z_2, z_3) be the output of Π' on those inputs. It holds that*

$$\Pr[y_i \neq z_i] \leq \mathsf{negl}(\delta),$$

where the probability is taken over the random coins of the parties and δ is the statistical security parameter.

Proof. From the assumed security of Π, it follows that there exists an ideal world adversary, corrupting P_3, corresponding to the real world adversary $\mathcal{A}_3$. We observe that the input of this ideal world adversary to the ideal functionality $\mathcal{F}$ has no influence on the outputs of the other parties. In particular, this means that P_2 has to receive output y_2 in the real world execution in the presence of $\mathcal{A}_3$ with an overwhelming (in δ) probability. Since P_2's view is identically distributed in the presence of $\mathcal{A}_3$ and $\mathcal{A}_1$, it follows that P_2's output in a real world execution of Π in the presence of $\mathcal{A}_1$ is also y_3.

It remains to show that P_3 obtains the correct output in an execution of Π in the presence of $\mathcal{A}_1$. From the definition of security against covert adversaries we know that the distribution of honest parties' outputs has to be statistically close in the real and ideal world for *all* possible inputs and thus this has to also be true when P_2's input $x_2 = \boldsymbol{c}$ is chosen uniformly at random. For corrupted party P_1 with input $x_1 = (\boldsymbol{b}_0, \boldsymbol{b}_1)$, let $\tilde{x}_1 = (\tilde{\boldsymbol{b}}_0, \tilde{\boldsymbol{b}}_1)$ be the input that the corresponding ideal world adversary sends to the ideal functionality $\mathcal{F}$. We observe that either $\boldsymbol{b}_0 = \tilde{\boldsymbol{b}}_0$ and $\boldsymbol{b}_1 = \tilde{\boldsymbol{b}}_1$ in which case P_3 obtains the right output for any input x_3 or

$$\Pr_{\boldsymbol{c}}[\langle \boldsymbol{b}_0, \boldsymbol{c}\rangle = \langle \tilde{\boldsymbol{b}}_0, \boldsymbol{c}\rangle \wedge \langle \boldsymbol{b}_1, \boldsymbol{c}\rangle = \langle \tilde{\boldsymbol{b}}_1, \boldsymbol{c}\rangle] \leq 1/2,$$

which would contradict our previous conclusion that P_3 obtains the correct output with an overwhelming probability.

□

Claim 13. *Fix any* $\boldsymbol{b}_0, \boldsymbol{b}_1, \hat{\boldsymbol{b}}_1, \boldsymbol{c} \leftarrow \{0,1\}^2$ *with* $\langle \boldsymbol{b}_1, \boldsymbol{c}\rangle = \langle \hat{\boldsymbol{b}}_1, \boldsymbol{c}\rangle$*, let* $\boldsymbol{x} = ((\boldsymbol{b}_0, \boldsymbol{b}_1), \boldsymbol{c}, 0)$*, and let* $\hat{\boldsymbol{x}} = \Big((\boldsymbol{b}_0, \hat{\boldsymbol{b}}_1), \boldsymbol{c}, 0\Big)$*. Let* $\mathsf{view}_2^{\boldsymbol{x}}$ *denote distribution of the view of party* P_2 *in executions of* Π' *with inputs* $\boldsymbol{x}$ *and let* $\mathsf{view}_2^{\hat{\boldsymbol{x}}}$ *be distribution of views corresponding to inputs* $\hat{\boldsymbol{x}}$*. Then*

$$\mathsf{view}_2^{\boldsymbol{x}} \approx_\epsilon \mathsf{view}_2^{\hat{\boldsymbol{x}}},$$

where $\epsilon = \mathsf{negl}(\delta)$*.*

Proof. Let us consider real world adversary $\mathcal{A}_3$, who this time also corrupts P_2, but instructs this party follow the protocol completely honestly. Since Π is secure against two corruptions, it follows that there exists a corresponding ideal world adversary $S_{\{2,3\}}$ that corrupts P_2 and P_3 in the ideal world and for all inputs produces views that are indistinguishable from a real execution with an overwhelming probability in δ. From the previous claims we also know that the honest parties do not abort and that all parties obtain the correct outputs. Let $\tilde{\boldsymbol{c}}$ and $\tilde{d}$ be the inputs that the ideal world adversary $S_{\{2,3\}}$ sends to $\mathcal{F}$.

We observe that for any correctly functioning $S_{\{2,3\}}$ it has to hold that $\tilde{\boldsymbol{c}} = c$ and $\tilde{d} = d$ with overwhelming probability. Assume this was not the case. Let $\boldsymbol{b}_0, \boldsymbol{b}_1 \in \{0,1\}^2$ be a uniformly random input of P_1. In the real world P_2 obtains $\langle \boldsymbol{b}_0, c\rangle$ and $\langle \boldsymbol{b}_1, c\rangle$ and P_3 obtains $\boldsymbol{b}_d$, which $S_{\{2,3\}}$ needs to simulate in the ideal world from knowing $\langle \boldsymbol{b}_0, \tilde{\boldsymbol{c}}\rangle, \langle \boldsymbol{b}_1, \tilde{\boldsymbol{c}}\rangle$ and $\boldsymbol{b}_{\tilde{d}}$. Now if $d \neq \tilde{d}$, then $S_{\{2,3\}}$ can simulate P_3's correct output with at best some constant probability smaller one and if $c \neq$

$\tilde{c}$, then it can only correctly simulate P_3's output with some constant probability smaller one. Let us now consider some arbitrary, but fixed $\boldsymbol{b}_0, \boldsymbol{b}_1, \hat{\boldsymbol{b}}_1, \boldsymbol{c} \in \{0,1\}^2$ with $\langle \boldsymbol{b}_1, \boldsymbol{c} \rangle = \langle \hat{\boldsymbol{b}}_1, \boldsymbol{c} \rangle$ and $d = 0$ as in the claim statement. From the discussion above we can conclude that the ideal world adversary will see $\langle \boldsymbol{b}_0, \boldsymbol{c} \rangle, \langle \boldsymbol{b}_1, \boldsymbol{c} \rangle$ and $\boldsymbol{b}_d$. This means that $S_{\{2,3\}}$'s simulation for inputs $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ will be identical and thus the corresponding real-world executions will be statistically close.

□

Claim 14. *Fix any* $\boldsymbol{b}_0, \hat{\boldsymbol{b}}_0, \boldsymbol{b}_1, \boldsymbol{c} \leftarrow \{0,1\}^2$ *with* $\langle \boldsymbol{b}_0, \boldsymbol{c} \rangle = \langle \hat{\boldsymbol{b}}_0, \boldsymbol{c} \rangle$. *For* $\boldsymbol{x} = ((\boldsymbol{b}_0, \boldsymbol{b}_1), \boldsymbol{c}, 1)$ *and* $\hat{\boldsymbol{x}} = \Big((\hat{\boldsymbol{b}}_0, \boldsymbol{b}_1), \boldsymbol{c}, 1\Big)$ *it holds that*

$$\mathsf{view}_2^{\boldsymbol{x}} \approx_\epsilon \mathsf{view}_2^{\hat{\boldsymbol{x}}},$$

where $\epsilon = \mathsf{negl}(\delta)$.

Proof. Claim can be proven completely symmetrically to the previous claim.

□

Claim 15. *Fix any* $\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{c} \leftarrow \{0,1\}^2$ *For* $\boldsymbol{x} = ((\boldsymbol{b}_0, \boldsymbol{b}_1), \boldsymbol{c}, 0)$ *and* $\hat{\boldsymbol{x}} = ((\boldsymbol{b}_0, \boldsymbol{b}_1), \boldsymbol{c}, 1)$ *it holds that*

$$\mathsf{view}_2^{\boldsymbol{x}} \approx_\epsilon \mathsf{view}_2^{\hat{\boldsymbol{x}}},$$

Proof. Claim can be proven symmetrically to the previous claim by just corrupting P_1 and P_2 instead of P_2 and P_3.

□

Claim 16. *Fix any* $\boldsymbol{b}_0, \hat{\boldsymbol{b}}_0, \boldsymbol{b}_1, \hat{\boldsymbol{b}}_1, \boldsymbol{c} \leftarrow \{0,1\}^2$ *with* $\langle \boldsymbol{b}_0, \boldsymbol{c} \rangle = \langle \hat{\boldsymbol{b}}_0, \boldsymbol{c} \rangle$ *and* $\langle \boldsymbol{b}_1, \boldsymbol{c} \rangle = \langle \hat{\boldsymbol{b}}_1, \boldsymbol{c} \rangle$. *For* $\boldsymbol{x} = ((\boldsymbol{b}_0, \boldsymbol{b}_1), c, 0)$ *and* $\hat{\boldsymbol{x}} = \Big((\hat{\boldsymbol{b}}_0, \hat{\boldsymbol{b}}_1), c, 1\Big)$ *we have that*

$$\mathsf{view}_2^{\boldsymbol{x}} \approx_{\epsilon'} \mathsf{view}_2^{\hat{\boldsymbol{x}}},$$

where $\epsilon' = 3\epsilon$ *is negligible in the statistical security parameter* δ.

Proof. From the previous three claims we get the following sequence of hybrids, which proves the statement:

$$\begin{aligned}
\mathsf{view}_2^{\boldsymbol{x}} &= \mathsf{view}_2^{((\boldsymbol{b}_0, \boldsymbol{b}_1), c, 0)} \\
&\approx_\epsilon \mathsf{view}_2^{\left((\boldsymbol{b}_0, \hat{\boldsymbol{b}}_1), c, 0\right)} \\
&\approx_\epsilon \mathsf{view}_2^{\left((\boldsymbol{b}_0, \hat{\boldsymbol{b}}_1), c, 1\right)} \\
&\approx_\epsilon \mathsf{view}_2^{\left((\hat{\boldsymbol{b}}_0, \hat{\boldsymbol{b}}_1), c, 1\right)} = \mathsf{view}_2^{\hat{\boldsymbol{x}}}.
\end{aligned}$$

□

At this point it will help our understanding to take stock of what we have shown so far. We have first shown that even if we cut the communication channel between P_1 and P_3, the protocol will not abort and correctly compute the desired functionality, which roughly corresponds to an oblivious transfer between P_1 and P_3. Moreover, we have shown that for any two inputs $(\boldsymbol{b}_0, \boldsymbol{b}_1)$ and $(\hat{\boldsymbol{b}}_0, \hat{\boldsymbol{b}}_1)$ of P_1, the view of P_2 remains the basically the same[7]. The fact that P_2's view is independent of P_1's input means that the view does not contain information about the input. Lastly, recall that we are considering information-theoretic protocols with oracle access two arbitrary two-party functionalities, which means that apart from the two-party functionalities, we are only given information-theoretic tools. In the last step of this proof, we will show that P_1 and P_3 cannot perform the desired oblivious transfer or, in other words, P_3's output cannot contain any information about either of P_1's inputs.

Claim 17. *Let $\boldsymbol{b}_0, \hat{\boldsymbol{b}}_0, \boldsymbol{b}_1, \hat{\boldsymbol{b}}_1 \leftarrow \{0,1\}^2$ be uniformly random and let $\boldsymbol{c} \leftarrow \{0,1\}^2$ and $d \leftarrow \{0,1\}$ be chosen arbitrary with $\langle \boldsymbol{b}_0, \boldsymbol{c} \rangle = \langle \hat{\boldsymbol{b}}_0, \boldsymbol{c} \rangle$ and $\langle \boldsymbol{b}_1, \boldsymbol{c} \rangle = \langle \hat{\boldsymbol{b}}_1, \boldsymbol{c} \rangle$. Let $\boldsymbol{x} = ((\boldsymbol{b}_0, \boldsymbol{b}_1), c, d)$ and $\hat{\boldsymbol{x}} = \Big((\hat{\boldsymbol{b}}_0, \hat{\boldsymbol{b}}_1), c, d\Big)$. Let $Z_3^{\boldsymbol{x}}$ be the distribution of outputs of P_3 in an execution of Π' on input vector $\boldsymbol{x}$ induced by random tapes chosen uniformly at random. Then it holds that*

$$\Pr[Z_3^{\boldsymbol{x}} = \boldsymbol{b}_d] \leq \Pr[Z_3^{\hat{\boldsymbol{x}}} = \boldsymbol{b}_d] + \mathsf{negl}(\delta).$$

Proof. Let us consider executions of Π', where the input vector is chosen uniformly at random to be either $\boldsymbol{x}$ or $\hat{\boldsymbol{x}}$ and the random tapes are chosen uniformly at random. Let V_2 be the distribution of P_2's view and let Z_3 be the distribution of P_3's output in such executions.

By Claim 16, we know that there exists an $\epsilon = \mathsf{negl}(\delta)$ such that

$$\frac{1}{2} \sum_v |\Pr[V_2 = v \mid X = \boldsymbol{x}] - \Pr[V_2 = v \mid X = \hat{\boldsymbol{x}}]| = \epsilon.$$

For each possible view v of P_2, we define $\epsilon_v = \Pr[V_2 = v \mid X = \boldsymbol{x}] - \Pr[V_2 = v \mid X = \hat{\boldsymbol{x}}]$. We observe that

$$\sum_v \epsilon_v \leq \sum_v |\epsilon_v| = 2\epsilon$$

and

$$\Pr[V_2 = v \mid X = \boldsymbol{x}] = \Pr[V_2 = v \mid X = \hat{\boldsymbol{x}}] + \epsilon_v.$$

Furthermore, since

$$\Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 = v]$$

and

$$\Pr[X = \boldsymbol{x} \mid V_2 = v]$$

[7] View may contain negligible traces of ~~nuts~~ information.

are conditionally independent for any view v of P_2, we observe that for the set $\mathbb{V}_2^*$ of views v^* of P_2 with

$$\Pr[X = \boldsymbol{x} \mid V_2 = v^*] \neq 0$$

and

$$\Pr[X = \hat{\boldsymbol{x}} \mid V_2 = v^*] \neq 0$$

it holds that

$$\begin{aligned}
\Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 = v^*, X = \boldsymbol{x}] &= \frac{\Pr[P_3(d) = \boldsymbol{b}_d, X = \boldsymbol{x} \mid V_2 = v^*]}{\Pr[X = \boldsymbol{x} \mid V_2 = v^*]} \\
&= \frac{\Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 = v^*] \cdot \Pr[X = \boldsymbol{x} \mid V_2 = v^*]}{\Pr[X = \boldsymbol{x} \mid V_2 = v^*]} \\
&= \frac{\Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 = v^*] \cdot \Pr[X = \hat{\boldsymbol{x}} \mid V_2 = v^*]}{\Pr[X = \hat{\boldsymbol{x}} \mid V_2 = v^*]} \\
&= \Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 = v^*, X = \hat{\boldsymbol{x}}].
\end{aligned}$$

Let us now analyze the probability of P_3 outputting the correct result on input vector $\boldsymbol{x}$ by combining our observations above and repeatedly applying Lemma 6, 7, 9 and Corollary 8.

$$\begin{aligned}
\Pr[P_3(d) = \boldsymbol{b}_d \mid X = \boldsymbol{x}] &= \Pr[V_2 \in \mathbb{V}_2^* \mid X = \boldsymbol{x}] \cdot \Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 \in \mathbb{V}_2^*, X = \boldsymbol{x}] \\
&\quad + \Pr[V_2 \notin \mathbb{V}_2^* \mid X = \boldsymbol{x}] \cdot \Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 \notin \mathbb{V}_2^*, X = \boldsymbol{x}] \\
&\leq \Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 \in \mathbb{V}_2^*, X = \boldsymbol{x}] + \mathsf{negl}(\delta) \\
&= \sum_v \Pr[P_3(d) = \boldsymbol{b}_d, V_2 = v \mid V_2 \in \mathbb{V}_2^*, X = \boldsymbol{x}] + \mathsf{negl}(\delta) \\
&= \sum_{v \in V_2^*} \Pr[P_3(d) = \boldsymbol{b}_d, V_2 = v \mid V_2 \in \mathbb{V}_2^*, X = \boldsymbol{x}] + \mathsf{negl}(\delta) \\
&= \sum_{v \in V_2^*} \frac{\Pr[P_3(d) = \boldsymbol{b}_d, V_2 = v \mid X = \boldsymbol{x}]}{\Pr[V_2 \in \mathbb{V}_2^* \mid X = \boldsymbol{x}]} + \mathsf{negl}(\delta) \\
&\leq \sum_{v \in V_2^*} \frac{\Pr[P_3(d) = \boldsymbol{b}_d, V_2 = v \mid X = \boldsymbol{x}]}{99/100} + \mathsf{negl}(\delta) \\
&= \frac{100}{99} \sum_{v \in V_2^*} \Pr[P_3(d) = \boldsymbol{b}_d, V_2 = v \mid X = \boldsymbol{x}] + \mathsf{negl}(\delta) \\
&= \frac{100}{99} \sum_{v \in V_2^*} \Pr[V_2 = v \mid X = \boldsymbol{x}] \cdot \Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 = v, X = \boldsymbol{x}] + \mathsf{negl}(\delta) \\
&= \frac{100}{99} \sum_{v \in V_2^*} (\Pr[V_2 = v \mid X = \hat{\boldsymbol{x}}] + \epsilon_v) \cdot \Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 = v, X = \hat{\boldsymbol{x}}] + \mathsf{negl}(\delta) \\
&= \frac{100}{99} \left(\sum_{v \in V_2^*} \Pr[V_2 = v \mid X = \hat{\boldsymbol{x}}] \cdot \Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 = v, X = \hat{\boldsymbol{x}}] \right.
\end{aligned}$$

$$
\begin{aligned}
&+ \sum_{v \in V_2^*} \epsilon_v \cdot \Pr[P_3(d) = \boldsymbol{b}_d \mid V_2 = v, X = \hat{\boldsymbol{x}}] \Bigg) + \mathsf{negl}(\delta) \\
&\leq \frac{100}{99} \left(\Pr[P_3(d) = \boldsymbol{b}_d \mid X = \hat{\boldsymbol{x}}] + \sum_{v \in V_2^*} \epsilon_v \right) + \mathsf{negl}(\delta) \\
&\leq \frac{100}{99} \Pr[P_3(d) = \boldsymbol{b}_d \mid X = \hat{\boldsymbol{x}}] + 2\epsilon + \mathsf{negl}(\delta) \\
&\leq \Pr[P_3(d) = \boldsymbol{b}_d \mid X = \hat{\boldsymbol{x}}] + 2\epsilon + \mathsf{negl}(\delta) + 1/99,
\end{aligned}
$$

which means that

$$
\Pr[P_3(d) = \boldsymbol{b}_d \mid X = \boldsymbol{x}] - \Pr[P_3(d) = \boldsymbol{b}_d \mid X = \hat{\boldsymbol{x}}] \leq 2\epsilon + \mathsf{negl}(\delta) + 1/99
$$

This means that the probability of P_3 returning $\boldsymbol{b}_d$ is effectively "independent" of the actual input that was used, which contradicts Claim 12. Therefore, the desired protocol with security a covert adversary who corrupts two parties cannot exists.

□

□

The following corollary immediately follows.

Corollary 18. *Let $\mathcal{F}_{\mathsf{2PC}}$ be an arbitrary two-party ideal functionality and let $\mathcal{F}_{\mathsf{BC}}$ be the ideal broadcast functionality. There exists a n-party functionality $\mathcal{F}$, which cannot be implemented with unconditional security against a covert adversary, who corrupts $t = \frac{2n}{3}$ parties, in the $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{BC}})$-hybrid model.*

B Secure Multiparty Computation

Passive adversaries. Security against passive adversaries is modeled by considering an environment $\mathcal{Z}$ that, in the real and ideal execution, picks the inputs of all parties. An adversary $\mathcal{A}$ gets access to views of the corrupted parties, but follows the protocol specification honestly. We consider the following ideal execution:

Inputs: Environment $\mathcal{Z}$ gets as input auxiliary information z and sends the vector of inputs $\bar{x} = (x_1, \ldots, x_n)$ to the ideal functionality $\mathcal{F}$.

Ideal functionality reveals inputs: If the ideal world adversary S sends $\mathsf{get\_inputs}$ to $\mathcal{F}$, then it gets back the inputs of all corrupted parties, i.e. all x_i, where $i \in I$.

Output generation: The ideal functionality computes $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and returns back y_i to each P_i. All honest parties output whatever they receive from $\mathcal{F}$. The ideal world adversary S outputs an arbitrary probabilistic polynomial-time computable function of the initial inputs of the corrupted parties, the auxiliary input z, and the messages received from the ideal functionality.

The joint distribution of the outputs of the honest parties and S in an ideal execution is denoted by $\mathsf{IDEAL}_\lambda[S(z), I, \mathcal{F}, \bar{x}]$.

Definition 9. *Protocol Π is said to securely compute $\mathcal{F}$ with security against passive adversaries in the $\mathcal{G}$-hybrid model if for every non-uniform probabilistic polynomial time adversary $\mathcal{A}$ in the real world, there exists a probabilistic polynomial time adversary S in the ideal world such that for all $\lambda \in \mathbb{N}$*

$$\left\{\mathsf{IDEAL}_\lambda[S(z), I, \mathcal{F}, \bar{x}]\right\}_{\bar{x}, z \in \{0,1\}^*} \equiv_c \left\{\mathsf{REAL}_\lambda[\mathcal{A}(z), I, \Pi, \mathcal{G}, \bar{x}]\right\}_{\bar{x}, z \in \{0,1\}^*}$$

B.1 Active Adversaries

Security with abort against active adversaries considers adversaries that may behave in an arbitrarily malicious fashion. We consider the following ideal execution:

Inputs: The honest parties send their inputs to the ideal functionality $\mathcal{F}$. All corrupted parties may either abort, send its prescribed input or an arbitrary different input to the ideal functionality.

Early Abort: If $\mathcal{F}$ receives abort from a corrupt party P_i instead of an input, then it sends abort to all the honest parties and terminates.

Ideal functionality answers adversary: The ideal functionality computes $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and sends it to S.

Ideal functionality answers honest parties: The adversary S either sends back $\mathsf{continue}$ or abort for a corrupted P_i. If the adversary sends $\mathsf{continue}$, then the ideal functionality returns y_i to each honest parties P_i. If the adversary sends abort, then the ideal functionality sends back abort to all honest parties.

Output generation: An honest party always outputs the message it obtained from $\mathcal{F}$. The corrupted parties output nothing. The adversary outputs an arbitrary probabilistic polynomial-time computable function of the initial inputs of the corrupted parties, the auxiliary input z, and the messages received from the ideal functionality.

The joint distribution of the outputs of the honest parties and S in an ideal execution is denoted by $\mathsf{IDEAL}_\lambda[S(z), I, \mathcal{F}, \bar{x}]$.

Definition 10. *Protocol Π is said to securely compute $\mathcal{F}$ with active security in the $\mathcal{G}$-hybrid model if for every non-uniform probabilistic polynomial time adversary $\mathcal{A}$ in the real world, there exists a probabilistic polynomial time adversary S in the ideal world such that for all $\lambda \in \mathbb{N}$*

$$\left\{\mathsf{IDEAL}^{\epsilon}_\lambda[S(z), I, \mathcal{F}, \bar{x}]\right\}_{\bar{x}, z \in \{0,1\}^*} \equiv_c \left\{\mathsf{REAL}_\lambda[\mathcal{A}(z), I, \Pi, \mathcal{G}, \bar{x}]\right\}_{\bar{x}, z \in \{0,1\}^*}$$

Identifiable abort. Apart from security against covert or active adversaries with abort, we will also consider security against covert or active adversaries with *identifiable* abort. Here, upon a protocol aborting, the honest parties unanimously agree and output an index i, which refers to one of the corrupt parties that caused the protocol to abort. A bit more formally, in the security notions above, the ideal functionality would return an additional index i whenever it sends abort to the honest parties, where P_i is a corrupted party. For a functionality $\mathcal{F}$, we write $[\mathcal{F}]^{\mathsf{ida}}$ to denote the corresponding ideal functionality with identifiable abort.

C Additional Definitions

C.1 Authenticated Secret-Sharing

A *authenticated* (threshold) secret sharing scheme $(\mathsf{share}, \mathsf{rec})$ is a standard secret sharing scheme with the following additional property:

Definition 11. *A secret sharing scheme $(\mathsf{share}, \mathsf{rec})$ for access structure $\mathbb{A}$ is said to be authenticated if the following holds for all possible secrets s, for all $n \in \mathbb{N}$, for any $C \subset [n]$ with $C \notin \mathbb{A}$, and any $B \in \mathbb{A}$*

$$\Pr\left[\begin{array}{rr} (s_1, \ldots, s_n) \leftarrow \mathsf{share}(s) & \\ S_C = \{s_i \mid i \in C\} & S'_C \neq S_C \\ S_B = \{s_i \mid i \in B \setminus C\} & : \mathsf{rec}(S'_C \cup S_B) = s \\ S'_C \leftarrow \mathcal{A}(S_C) & \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where probability is taken over the random coins of the sharing algorithm share and coins of the adversary $\mathcal{A}$.

Such secret sharing schemes can easily be constructed from standard secret sharing schemes in combination with MACs or signatures.

C.2 Aggregate Signatures

An aggregate signature scheme is a tuple $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{AggVer})$, where $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ constitutes a standard existentially signature scheme, which is equipped with the following additional algorithms:

Agg: The aggregation algorithm takes a list of message/signature pairs $((m_1, \sigma_1), \ldots, (m_n, \sigma_n))$ as input and outputs an aggregate signature σ.

AggVer: The aggregate verification algorithm takes a list of public keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$, a signature σ, and a list of messages $m_1, \ldots, m_n$.

Definition 12 (Existential Unforgeability in the Chosen-Key Model). *An aggregate signature scheme* $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{AggVer})$ *is said to be secure in the aggregate chosen-key model if for any PPT adversary* $\mathcal{A}$ *it holds that*

$$\Pr\left[\begin{array}{r} \mathsf{vk}_1 \leftarrow \mathsf{Gen}(\lambda) \\ (\mathsf{vk}_2, \ldots, \mathsf{vk}_n, \sigma, m) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{vk}_1) \end{array} : \begin{array}{r} \mathsf{AggVer}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, \sigma, m) = 1 \\ m \notin Q \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathcal{O}(\cdot)$ *is the oracle that takes messages as input and outputs signatures under key* sk_1 *and* Q *is the set of messages that were queried to this oracle.*

C.3 Correlation robust hash function

Definition 13 (Correlation robust hash function). *We say that a hash function* $\mathsf{H}_{\mathsf{cr}} : \{0,1\}^\lambda \to \{0,1\}$ *is (strongly)* correlation robust *if for all* $t_1, \ldots, t_m \in \{0,1\}^\lambda$ *chosen by an adversary, it holds that the distribution*

$$\left\{(\mathsf{H}_{\mathsf{cr}}(t_1 \oplus R), \ldots, \mathsf{H}_{\mathsf{cr}}(t_m \oplus R)) \mid R \leftarrow \{0,1\}^\lambda\right\}$$

is computationally indistinguishable from the uniform distribution.

C.4 Useful Ideal Functionalities

Figure C.1: $\mathcal{F}_{\mathsf{FLIP}}$

The functionality interacts with parties $P_1, \ldots, P_n$.

- **Coin Flip:** If party P_i sends $(\mathsf{coinFlip}, \mathsf{id})$ to the ideal functionality, it stores $(\mathsf{id}, \mathsf{coinFlip}, P_i)$ in memory. If $(\mathsf{id}, \mathsf{coinFlip}, P_i)$ was already stored, then ignore the command. Once, for an identifier id, one such entry from every party exists. The functionality picks a random value r_{id} and sends $(\mathsf{coinFlip}, \mathsf{id}, r_{\mathsf{id}})$ to all parties
- **Commit Phase:** If party P_i sends $(\mathsf{comFlip}, \mathsf{id})$ to the ideal functionality, it stores $(\mathsf{id}, \mathsf{comFlip}, P_i)$ in memory. If $(\mathsf{id}, \mathsf{comFlip}, P_i)$ was already stored, then ignore the command. Once, for an identifier id, one such entry from every party exists. The functionality picks a random value r_{id}, stores $(\mathsf{id}, r_{\mathsf{id}})$ in memory, and sends $(\mathsf{com}, \mathsf{id})$ to all parties
- **Open to all:** If party P_i sends $(\mathsf{openAll}, \mathsf{id})$, store $(\mathsf{openAll}, \mathsf{id}, i)$ in memory. If one such entry from every party exists and if there exists a corresponding entry $(\mathsf{id}, r_{\mathsf{id}})$, then send $(\mathsf{openAll}, r_{\mathsf{id}})$ to all parties.
- **Open to one:** If party P_i sends $(\mathsf{openOne}, \mathsf{id}, j)$, store $(\mathsf{openOne}, \mathsf{id}, i, j)$ in memory. If one such entry from every party

exists and if there exists a corresponding entry $(\mathsf{id}, r_{\mathsf{id}})$, then send $(\mathsf{openOne}, r_{\mathsf{id}})$ to P_j.

Figure C.2: $\mathcal{F}_{\mathsf{COM}}$

The functionality interacts with parties $P_1, \ldots, P_n$.

- **Commit Phase:** If party P_i sends $(\mathsf{commit}, \mathsf{id}, m)$ to the ideal functionality, it checks whether a tuple $(\mathsf{id}, *, *)$ was already stored in memory. If yes, then the ideal functionality ignores the command. Otherwise it stores (id, i, m) in memory and sends $(\mathsf{receipt}, \mathsf{id})$ to all other parties.
- **Open:** If party P_i sends $(\mathsf{open}, \mathsf{id})$, the functionality checks, whether there is a corresponding entry (id, i, m) stored in memory. If this is the case, the functionality returns $(\mathsf{open}, \mathsf{id}, m)$ to all other parties.

Figure C.3: $\mathcal{F}_{\mathsf{CHECK}}$

The functionality interacts with parties $P_1, \ldots, P_n$.

- **Input Phase:** If party P_i sends $(\mathsf{check}, \mathsf{id}, K_{i,1}, \ldots, K_{i,k})$, it checks whether a tuple $(\mathsf{id}, *, *)$ was already stored in memory. If yes, then the ideal functionality ignores the command. Otherwise it stores $(\mathsf{id}, i, K_{i,1}, \ldots, K_{i,k})$ in memory and sends $(\mathsf{receipt}, \mathsf{id})$ to all other parties.
- **Output Phase:** Once every party P_i provided input to the functionality, it picks a uniformly random permutation γ over $[k]$. It uses authenticated secret sharing to share

$$\boldsymbol{s} = \left(\begin{pmatrix} K_{1,\gamma(1)}, \ldots, K_{n,\gamma(1)} \\ \vdots \\ K_{1,\gamma(k-1)}, \ldots, K_{n,\gamma(k-1)} \end{pmatrix}, \gamma \right)$$

into shares $(s_1, \ldots, s_n)$ and sends back share s_i to party P_i.

D Additional BMR Section

D.1 Preprocessing for BMR with Public Verifiability

Here we describe the preprocessing functionality, and our protocol for realising it with passive security. The functionality, shown in Figure D.1, essentially follows the standard description of BMR preprocessing with the free-XOR technique, as

used in previous works [8, 23]. For each wire w, party P_i obtains a random key K_w^i, where the 'zero key' for that wire is defined to be $(K_w^1, \ldots, K_w^n)$. Each party also has a fixed offset R^i to allow free-XOR, so that the 'one key' for each wire is $(K_w^1 \oplus R^1, \ldots, K_w^n \oplus R^n)$. We also use the point-and-permute technique [7], so that each wire is additionally associated with a mask $\lambda_w \in \{0,1\}$, which is XOR shared between the parties.

The functionality then outputs random shares of the garbled circuit generated with those keys, where the corrupt parties can choose their own shares and keys (and honest outputs are reverse-sampled accordingly).

Figure D.1: The Preprocessing Functionality $\mathcal{F}_{\mathsf{bmr}}$

Let H be a hash functions and com a commitment scheme.
C is a boolean circuit with set of wires W, W_{out} its set of output wires, W_{in} its set of input wires, of which W_{in}^i are those for inputs from P_i, and G its set of gates. Each gate is indexed by a unique identifier g, of which we denote the subsets of XOR and AND gates by XOR and AND respectively.

1. Sample a global difference $R^i \leftarrow \{0,1\}^\lambda$, for each $i \in [n]$
2. For each wire $w \in W_{\mathsf{in}}$, sample the keys $K_w^i \leftarrow \{0,1\}^\lambda$, for $i \in [n]$, and the mask $\lambda_w \leftarrow \{0,1\}$.
3. For each $g \in G$ with input wires u, v and output wire w, in topological order:
   - If $g \in \mathsf{XOR}$, let $\lambda_w = \lambda_u \oplus \lambda_v$ and $K_w^i = K_u^i \oplus K_v^i$, for $i \in [n]$.
   - If $g \in \mathsf{AND}$:
     (a) Sample $K_w^i \leftarrow \{0,1\}^\lambda$, for $i \in [n]$, and $\lambda_w \leftarrow \{0,1\}$.
     (b) For $a, b \in \{0,1\}$, let $c = c(a,b) = (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \oplus \lambda_w$. Compute the four entries of the garbled gate as:

$$\widetilde{g}_{a,b} = \bigoplus_{i=1}^{n} \mathsf{H}(g, i, K_u^i \oplus aR^i, K_v^i \oplus bR^i) \oplus (K_w^1 \oplus cR^1, \ldots, K_w^n \oplus cR^n)$$

4. Let GC be the concatenation of $\widetilde{g}_{a,b}$, for each $g \in G$ and $a, b \in (0,1)^2$. Sample random shares GC^i such that $\mathsf{GC} = \bigoplus_i \mathsf{GC}^i$.

Output: Send to each party P_i the GC share GC^i, keys R^i and $\{K_w^i\}_{w \in W}$, and the wire masks $\{\lambda_w\}_{w \in W_{\mathsf{in}}^i \cup W_{\mathsf{out}}}$.
Corrupt Parties: If any P_i is corrupt, instead let $\mathcal{A}$ choose all its outputs above. Recompute GC and re-sample the honest parties' shares to be consistent with this.

Secret-sharing notation. Before describing our protocol, we introduce some notation for additive secret sharing. We write

$$[x^i R^j]_{ij} := ((x^i, s^i), (R^j, s^j))$$

to denote that parties P_i and P_j respectively hold $x^i \in \{0,1\}, s^i \in \{0,1\}^\lambda$ and $R^j, s^j \in \{0,1\}^\lambda$, such that $s_i \oplus s_j = x^i R^j$. Given two such sharings $[x^i R^j]_{ij}$ and $[y^i R^j]_{ij}$, the parties can locally XOR their shares to obtain $[z^i R^j]_{ij}$, where $z^i = x^i \oplus y^i$.

We also sometimes use n-party secret sharing, where we simply write

$$[x] := (x^1, \ldots, x^n)$$

to mean that each P_i holds x^i, and $x = \bigoplus_i x^i$. We overload the $\oplus$ operator on $[\cdot]$-shared values to mean local XOR of shares.[8]

Given a set of two-party sharings $[x^i R^j]_{ij}$, for every $i \neq j$, where P_j also holds a share x^j, the parties can locally convert this to a valid n-party sharing of xR^j, where $x = \bigoplus_i x^i$, as follows:

- Let P_i and P_j's shares of $x^i R^j$ be s^i and $t^{j,i}$, respectively.
- P_i, for $i \neq j$, outputs $z^i = s^i$.
- P_j outputs $z^j = x^j R^j \oplus \bigoplus_{i \neq j} t^{j,i}$.

We have $\bigoplus_{i=1}^n z^i = x^j R^j \oplus \bigoplus_{i \neq j} (s^i \oplus t^{j,i}) = xR^j$, as required.

Basic functionalities. As building blocks, the BMR preprocessing protocol uses a correlated oblivious transfer functionality $\mathcal{F}_{\mathsf{cot}}$ (Figure D.2), which we use to produce the $[x^i R^j]_{ij}$ sharings securely. We also use $\mathcal{F}_{\mathsf{zero}}$, and random zero-sharing functionality, which can easily be instantiated non-interactively after distributing PRF keys to the parties. Finally, we use a secret-shared multiplication functionality in $\mathbb{F}_2$, given by $\mathcal{F}_{\mathsf{mult}}$ (Figure D.4). This can be instantiated using pairwise oblivious transfer in a standard way, where the cost is $n(n-1)$ random OTs and $2n(n-1)$ bits of communication per multiplication.

Figure D.2: Functionality $\mathcal{F}_{\mathsf{cot}}$

The functionality operates between a sender, P_S and a receiver, P_R.

Initialize: Upon receiving $(init, \Delta)$, where $\Delta \in \{0,1\}^\lambda$ from P_S and $(init)$ from P_R, store Δ.

OT: Upon receiving $(OT, x_1, \ldots, x_m)$ from P_R, where $x_i \in \{0,1\}$, and (OT) from P_S, do the following:

- Sample $t_i \in \{0,1\}^\lambda$, for $i \in [m]$. If P_R is corrupted then wait for $\mathcal{A}$ to input t_i.
- Compute $q_i = t_i + x_i \cdot \Delta$, for $i \in [m]$.
- If P_S is corrupted then wait for $\mathcal{A}$ to input $q_i \in \{0,1\}^\lambda$ and recompute $t_i = q_i + x_i \cdot \Delta$.
- Output t_i to P_R and q_i to P_S, for $i \in [m]$.

[8] The parties can also perform addition by a constant, i.e. $[x] \oplus c$, by having a fixed party, say P_1, add the constant c to its share of x.

Figure D.3: Functionality $\mathcal{F}_{\mathsf{zero}}$

On input $\ell \in \mathbb{N}$ from each party, the functionality samples $z^1, \ldots, z^{n-1} \in \{0,1\}^\ell$, lets $z^n = \bigoplus_{i=1}^{n-1} z^i$, and outputs z^i to party P_i.

Figure D.4: Functionality $\mathcal{F}_{\mathsf{mult}}$

On input $x^i, y^i \in \{0,1\}$ from each party P_i, the functionality computes $x = \bigoplus_i x_i, y = \bigoplus y_i, z = x \cdot y$ and samples random z_i such that $z = \bigoplus z_i$. Output z_i to party P_i.

Optimized BMR preprocessing. We now describe our optimized passively secure preprocessing protocol.

Our protocol uses *correlated OT* to allow parties to generate additive (XOR) shares of the garbled circuit. This functionality receives as input a string $R^j \in \{0,1\}^\lambda$ from one party P_j, a bit $x^i \in \{0,1\}$ from another party P_i, and outputs the random two-party XOR sharing of $x^i R^j$, denoted $[x^i R^j]_{ij}$. This can be seen as an OT on the sender's messages $(q, q \oplus R^j)$, where q is the share output to P_j. Note that the string R^j is the same for every invocation of $\mathcal{F}_{\mathsf{cot}}$, and will correspond to P_j's offset for the free-XOR technique.

Previous passively secure protocols [8, 12] used $4n(n-1)$ OTs per AND gate, whereas we show how to reduce this to $3n(n-1)$. We first briefly recap the protocol of [8]. At each AND gate with input wires u, v and output wire w, the parties have random shared wire masks $\lambda_u, \lambda_v, \lambda_w \in \{0,1\}$, and need to obtain shares of the values

$$((\lambda_u \oplus a) \cdot (\lambda_v \oplus b) \oplus \lambda_w) \cdot R^j$$

for each $(a,b) \in (0,1)^2$ and $j \in [n]$, where R^j is the secret free-XOR offset known to P_j.

[8] observed that given n-party shares of $\lambda_u R^j, \lambda_v R^j$ and $(\lambda_{uv} \oplus \lambda_w) R^j$, where $\lambda_{uv} := \lambda_u \lambda_v$, all $4n$ shares above can be computed locally. This requires 4 sets of OTs between every pair of parties, first to obtain shares of λ_{uv}, and then the 3 products with R^j.

We take a different approach, where instead of garbling each AND gate separately, we start at the *input wires*. We will first ensure that for each wire w that is either an input wire, or the output wire of an AND gate, the parties get the sharings $[\lambda_w R^j]$; this costs $n(n-1) \cdot (|W_{\mathsf{in}}| + |\mathsf{AND}|)$ calls to $\mathcal{F}_{\mathsf{cot}}$ (on random inputs). By passing these sharings through all XOR gates and adding them accordingly, the parties can now *locally* obtain sharings of $\lambda_u R^j, \lambda_v R^j, \lambda_w R^j$ for each AND gate with wires u, v, w.

Next, the parties need two more sets of $n(n-1)$ OTs for each gate: first, to multiply the sharings of λ_u with λ_v, to obtain $[\lambda_{uv}]$, the parties call $\mathcal{F}_{\mathsf{mult}}$, which costs $n(n-1)$ random OTs and $2n(n-1)$ bits of communication. Secondly, we

need a further $n(n-1)$ calls to $\mathcal{F}_{\mathsf{cot}}$ (on chosen inputs) to multiply these with each R^j, allowing the correct shares to be computed.

In all, this gives a cost of $3n(n-1)$ OTs and $3n(n-1)$ bits of interaction per AND gate, plus $n(n-1)$ OTs for each input wire, where we have counted the number random OTs, and add an extra bit of communication in case this needs to be converted to a chosen-input OT.

Figure D.5: Protocol Π_{bmr}

Let H be a hash function.
C is a boolean circuit with set of wires W, W_{out} its set of output wires, W_{in} its set of input wires, of which W^i_{in} are those for inputs from P_i, and G its set of gates, of which we denote the subsets of XOR and AND gates by XOR and AND respectively. Each gate is indexed by a unique identifier g.

1. Each party P_i samples a global difference $R^i \leftarrow \{0,1\}^\lambda$. For each (i,j) with $i \neq j$, P_i and P_j call $\mathcal{F}_{\mathsf{cot}}$, where P_i inputs $(init, R^i)$.
2. For each input wire $w \in W_{\mathsf{in}}$, and each wire w that is an output of an AND gate:
   (a) Each party P_i samples a wire mask share $\lambda^i_w \leftarrow \{0,1\}$ and a key $K^i_w \leftarrow \{0,1\}^\lambda$.
   (b) For each $i \in [n]$, $j \neq i$, P_i and P_j call $\mathcal{F}_{\mathsf{cot}}$, where P_i inputs λ^i_w, to obtain the sharing $[\lambda^i_w R^j]_{ij}$.
3. For each gate $g \in \mathsf{XOR}$ with input wires $\{u,v\}$ and output wire w:
   (a) Each party P_i computes $K^i_w = K^i_u \oplus K^i_v$.
   (b) All parties compute the shares $[\lambda_w] = [\lambda_u] \oplus [\lambda_v]$, and each pair (P_i, P_j) computes
   $$[\lambda^i_w R^j]_{ij} = [\lambda^i_u R^j]_{ij} \oplus [\lambda^i_v R^j]_{ij}$$
4. For each gate $g \in \mathsf{AND}$ with input wires $\{u,v\}$ and output wire w:
   (a) The parties call $\mathcal{F}_{\mathsf{mult}}$ with input $[\lambda_u]$, and $[\lambda_v]$ to obtain shares $[\lambda_{uv}] := [\lambda_u \cdot \lambda_v]$.
   (b) Each pair (P_i, P_j) calls $\mathcal{F}_{\mathsf{cot}}$, where P_i inputs λ^i_{uv}, to obtain a sharing $[\lambda^i_{uv} R^j]_{ij}$.
   (c) The parties locally convert their pairwise sharings to n-party sharings $[\lambda_{uv} \cdot R^j]$, and do the same for $\lambda_u, \lambda_v, \lambda_w$, obtaining $[\lambda_u \cdot R^j]$, $[\lambda_v \cdot R^j]$ and $[\lambda_w \cdot R^j]$.
   (d) Using these sharings, the parties compute the sharings:
   $$[Z_{g,j,a,b}] := [((\lambda_u \oplus a) \cdot (\lambda_v \oplus b) \oplus \lambda_w) \cdot R^j] \quad \text{for } j \in [n], (a,b) \in (0,1)^2$$
   (e) Denote by $Z^i_{g,j,a,b}$ party P_i's share of the above. P_i computes
   $$\begin{aligned} \widetilde{g}^i_{a,b} = &\mathsf{H}(i, g, K^i_u \oplus aR^i, K^i_v \oplus bR^i) \oplus \\ &(Z^i_{g,1,a,b}, \ldots, Z^i_{g,n,a,b}) \oplus \\ &(0, \ldots, K^i_w, \ldots, 0), \quad \text{for } a,b \in \{0,1\}^2 \end{aligned}$$

(f) The parties call $\mathcal{F}_{\mathsf{zero}}$ on input $\ell = 4n\lambda|\mathsf{AND}|$, so each P_i obtains $Z^i \in \{0,1\}^\ell$. Let $\widetilde{\mathsf{GC}}^i$ be the concatenation of all $\widetilde{g}^i_{a,b}$, for $a, b \in \{0,1\}^2$ and $g \in \mathsf{AND}$.

(g) P_i defines $\mathsf{GC}^i = \widetilde{\mathsf{GC}}^i \oplus Z^i$.

5. For each $w \in W_{\mathsf{out}}$, each party P_i broadcasts λ^i_w, and the parties reconstruct $\lambda_w = \bigoplus_i \lambda^i_w$.
6. For each $w \in W^j_{\mathsf{in}}$, for $j \in [n]$, each P_i sends λ^i_w to P_j. P_j reconstructs $\lambda_w = \bigoplus_i \lambda^i_w$.
7. Party P_i outputs the garbled circuit share GC^i, the keys R^i and $\{K^i_w\}_{w \in W}$, and the input/output wire masks $\{\lambda_w\}_{w \in W^i_{\mathsf{in}} \cup W_{\mathsf{out}}}$.

Lemma 19. *The protocol Π_{bmr} securely realizes functionality $\mathcal{F}_{\mathsf{bmr}}$ with passive security, in the $(\mathcal{F}_{\mathsf{cot}}, \mathcal{F}_{\mathsf{zero}}, \mathcal{F}_{\mathsf{mult}})$-hybrid model.*

Proof. (sketch.) Note that the parties only ever interact with the ideal functionalities $\mathcal{F}_{\mathsf{cot}}, \mathcal{F}_{\mathsf{zero}}, \mathcal{F}_{\mathsf{mult}}$, until the very last step when they reconstruct the wire masks λ_w. Security therefore follows from the correctness of the computations, which can be seen by inspection, or following the same analysis from previous works such as [8]. □

D.2 Preprocessing for BMR with Identifiable Abort

To achieve identifiable abort, we modify the preprocessing functionality as discussed in Section 6, by having each party be committed to its shares of the garbled circuit and wire keys. For this, we use a unanimously identifiable commitment scheme, from [26], given by the following two algorithms.

$\mathsf{UIC.com}(x, n)$: On input $x \in \mathbb{F}$ and $n \in \mathbb{N}$:

- Sample a random, degree-$(n+1)$ polynomial $p(X)$ over $\mathbb{F}$ such that $P(0) = x$.
- For each $i \in [n]$, sample random $x_i \in \mathbb{F}$ and let $y_i = P(x_i)$.
- Output $c_i := (x_i, y_i)$ to party P_i, and $p(X)$ to the committer.

$\mathsf{UIC.dec}(c_i = (x_i, y_i), p(X))$: If $P(x_i) \neq y_i$, reject. Otherwise, output the message $x = P(0)$.

The above scheme assumes the messages lie in the finite field $\mathbb{F}$. In our protocol, we instead use a collision-resistant hash function, so we can support large messages whilst still having compact commitments. This means the scheme is no longer information-theoretically binding, but still satisfies the properties we need.

As long as $\mathbb{F}$ is large enough, the scheme UIC is binding, and also guarantees agreement among all n receivers, i.e. if one receiver does not accept the decomittment then all other receivers will do the same. We note that UIC also satisfies the following equivocation property. Given an arbitrary set of commitments $\{(x_i, y_i)\}_{I \subset [n]}$, and a target message (or message hash) m, we can find

a valid decommitment polynomial $p(X)$ such that $\mathsf{UIC.dec}((x_i, y_i), p(X)) = m$. This is true because we are given up to $n+1$ points, so can easily sample a random degree-$n+1$ polynomial with these evaluations. We finally note that given the decomitment and the message, it is possible to sample a valid commitment, which follows from the description of $\mathsf{UIC.com}$.

The BMR preprocessing functionality is then modified to output UIC commitments, with the extra step shown in Figure D.6. Notice that the functionality allows corrupt parties to choose their own outputs, and then reverse-samples the honest parties' outputs accordingly, from the conditional distribution. This is possible thanks to the equivocability of the information-theoretic UIC scheme, and ensures that the functionality fits the requirements of our covert compiler.

Figure D.6: Additional Step for Functionality $\mathcal{F}_{\mathsf{bmr\text{-}com}}$

After generating the garbled circuit shares GC^i as described in $\mathcal{F}_{\mathsf{bmr}}$, run the following additional step:

- For $i \in [n]$ and $w \in W_{\mathsf{in}}$, generate commitments and opening information

$$(D^i_{\mathsf{GC}}, \{C^{j,i}_{\mathsf{GC}}\}_{j\neq i}) = \mathsf{UIC.com}(\mathsf{GC}^i, n-1)$$
$$(D^i_{w,0}, \{C^{j,i}_{w,0}\}_{j\neq i}) = \mathsf{UIC.com}(K^i_w, n-1)$$
$$(D^i_{w,1}, \{C^{j,i}_{w,1}\}_{j\neq i}) = \mathsf{UIC.com}(K^i_w \oplus R^i, n-1)$$

As well as the previous outputs, send to P_i its decommitments $D^i_{\mathsf{GC}}, \{D^i_{w,0}, D^i_{w,1}\}_{w\in W_{\mathsf{in}}}$, and commitments $\{C^{i,j}_{\mathsf{GC}}, C^{i,j}_{w,0}, C^{i,j}_{w,1}\}_{i\in[n], j\neq i, w\in W_{\mathsf{in}}}$.

Corrupt Parties: For corrupt P_i, as well as allowing $\mathcal{A}$ to choose P_i's private randomness (as in the previous $\mathcal{F}_{\mathsf{bmr}}$ functionality), $\mathcal{A}$ may also choose the decommitments and commitments $D^i, C^{i,j}$ given to P_i. Given $D^i, C^{i,j}$ from P_i (for each corrupted P_i), reverse-sample the honest parties outputs consistently as follows: (a) Use D^i to generate the corresponding honest parties' commitments (just following the steps of $\mathsf{UIC.com}$); (b) Use the the equivocation algorithm for UIC in order to generate a decommitment for the honest P_j corresponding to $C^{i,j}$.

Preprocessing Protocol. In our protocols, we cannot have $\mathsf{UIC.com}$ be run by a single party, since otherwise they would be able to forge openings. Instead, we use the functionality $\mathcal{F}^{\mathbb{F},n}_{\mathsf{UIC}}$, which receives as input a message x from party P_j, and outputs commitments to all other parties, and the decommitment to P_j. In practice, $\mathcal{F}^{\mathbb{F},n}_{\mathsf{UIC}}$ can be realised with a passively secure protocol for oblivious polynomial evaluation, run between P_j and every other party. For instance, we can use a simple protocol based on additively homomorphic encryption, where P_j encrypts each coefficient of $P(X)$ and sends this to all other parties, who homomorphically evaluate $P(x)$ on a random point.

Given this, it is straightforward to modify the Π_{bmr} protocol to also output the UIC commitments, realizing $\mathcal{F}_{\mathsf{bmr\text{-}com}}$. Each party simply calls $\mathcal{F}_{\mathsf{UIC}}^{\mathbb{F},n}$ on input its share GC^i, and keys $K^i_{w,0}, K^i_{w,1}$, so that all parties obtain the correct commitments.

Figure D.7: Functionality $\mathcal{F}_{\mathsf{UIC}}^{\mathbb{F},n}$

The functionality operates with parties $P_1, \ldots, P_n$, and works over a finite field $\mathbb{F}$, and also uses a collision-resistant hash function $\mathsf{H}_{\mathsf{cr}} : \{0,1\}^* \to \mathbb{F}$. On input (com, j, x) from party P_j, where $x \in \mathbb{F}$, and (com, i) from all other parties:

1. Sample a random, degree-n polynomial $p(X)$ over $\mathbb{F}$ such that $P(0) = \mathsf{H}_{\mathsf{cr}}(x)$.
2. For each $i \in [n] \setminus \{j\}$, sample a random $x_i \in \mathbb{F}$ and let $y_i = P(x_i)$.
3. Output $c_i := (x_i, y_i)$ to party P_i, for $i \neq j$, and $p(X)$ to P_j.

D.3 Online Phase for BMR

Figure D.8: Protocol: Π^{on}

Let H be a 2-circular correlation robust hash function. C is a boolean circuit with set of wires W, W_{out} its set of output wires, W_{in} its set of input wires, of which W^i_{in} are those for inputs from P_i, and G its set of gates. Each gate is indexed by a unique identifier g, of which we denote the subsets of XOR and AND gates by XOR and AND respectively.
The parties execute the following commands in sequence:

Reconstruct Garbled Circuit:

1. Call $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ so that each party P_i receives the share GC^i, UIC commitments $\{C^{i,j}_{\mathsf{GC}}\}_{j\neq[n]}, \{C^{i,j}_{w,0}, C^{i,j}_{w,1}\}_{j\neq[n],w\in W_{\mathsf{in}}}$, input/output wire masks $\{\lambda_w\}$, as well as their wire keys and decommitments.
2. For each input wire $w \in W^i_{\mathsf{in}}$, party P_i computes and broadcasts $\Lambda_w = x^i_w \oplus \lambda_w$, where x^i_w is P_i's input for that wire.
3. For each $w \in W_{\mathsf{in}}$, each party P_i then broadcasts the key K^i_{w,Λ_w} associated to Λ_w, as well as its corresponding decommitment. Every other P_j verifies this is consistent with the commitment $C^{j,i}_{w,\Lambda_w}$.
4. Each party P_i broadcasts GC^i and its randomness for the commitment. The parties reconstruct $\mathsf{GC} = \bigoplus \mathsf{GC}^i$.
5. If any opening of a commitment from some P_i above fails, the parties output (abort, P_i).

Evaluate Garbled Circuit: For each gate $g \in G$, with input wires u, v and output wire w, in topological order:

1. If $g \in \mathsf{AND}$, compute:

$$(K^1_{w,\Lambda_w}, \cdots, K^n_{w,\Lambda_w}) = \tilde{g}_{\Lambda_u,\Lambda_v} \oplus \bigoplus_{i=1}^{n} \left(\mathsf{H}(g, i, K^i_{u,\Lambda_u}, K^i_{v,\Lambda_v})\right)$$

and let Λ_w be the bit such that $K^i_{w,\Lambda_w} = K^i_w \oplus (\Lambda_w \cdot R^i)$ (for party P_i, who received R^i, K^i_w in the preprocessing).

2. If $g \in \mathsf{XOR}$, compute $\Lambda_w = \Lambda_u \oplus \Lambda_v$ and $K^i_{w,\Lambda_w} = K^i_{u,\Lambda_u} \oplus K^i_{v,\Lambda_v}$ for $i \in [n]$.

In Figure D.8 we present our online phase that use for BMR with identifiable abort. Note that for the case of public verifiability (without identifiability), we can just use the same online phase as previous works [8, 23]. The identifiable protocol, shown in Figure D.8, simply follows the standard online phase of passive BMR, with the difference that whenever a party broadcasts input wire keys, or a share of the garbled circuit, it also opens its commitment to ensure this is done correctly. This leads to an online phase that is *actively secure with identifiable abort*, when the preprocessing is realized by an ideal functionality.

Lemma 20. *Let* H *be a 2-circular correlation robust hash function, and suppose* UIC *is a secure unanimously identifiable commitment scheme. Then, protocol* Π^{on} *securely realizes* $\mathcal{F}_{\mathsf{on}}$ *with active security and identifiable abort in the* $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$*-hybrid model.*

Proof. (sketch) The simulation is identical to the proof of the online phase from [23], with the only difference that in our case, the parties are also committed to their shares and keys, so if any corrupt P_i sends an invalid opening, the simulator can simply send (abort, i) to the ideal functionality. When simulating the honest parties' garbled circuit shares, we can use the equivocability of the commitment scheme so that this can be done in a way consistent with the actual output of the circuit. Because of the security of UIC, we are guaranteed that in case of abort, all honest parties will unanimously agree upon the same cheating party, i.e., all the honest parties can detect invalid decommitments unanimously. □

D.4 Combining the Online Phase with Covertly Secure Preprocessing

Lemma 21. *Suppose* Π^{on} *securely realizes* $\mathcal{F}_{\mathsf{on}}$ *with active security and identifiable abort in the* $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$*-hybrid model, and that* $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$ *securely realizes* $\mathcal{F}^{\mathcal{D}}_{\mathsf{corr}}$ *with covert security and identifiable abort. Then, the combined protocol* $(\Pi^{\mathsf{on}}, [\Pi_{\mathsf{corr}}]^{\mathsf{cov}})$ *securely realizes* $\mathcal{F}_{\mathsf{on}}$ *with covert security and identifiable abort.*

Proof. In order to prove the above statement we would like to exhibit a simulator S that produces a trasnscript in the ideal world (acting as the adversary) that is indistinguishable from the one produced form the real world execution of $(\Pi^{\mathsf{on}}, [\Pi_{\mathsf{corr}}]^{\mathsf{cov}})$.

We first observe that from the covert security (resp., active security) with identifiable abort implies that there exists a simulator $[S_{\mathsf{corr}}]^{\mathsf{cov}}$ for $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$ (resp. S_{on} for Π^{on}).

We will now briefly describing S_{corr} which interacts with adversary $\mathcal{A}$. S_{corr} runs $[S_{\mathsf{corr}}]^{\mathsf{cov}}$ during the execution of $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$ in order to interact with $\mathcal{A}$, simulating for him the functionality $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$. Indeed, S_{corr} acts as a proxy for the messages exchanged between S_{corr} in particular:

1. If $[S_{\mathsf{corr}}]^{\mathsf{cov}}$ sends the message $(\mathsf{corrupted}, i)$ to S_{corr} (simulating $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$) the simulator S_{corr} forwards it to $\mathcal{F}_{\mathsf{on}}$.
2. If $[S_{\mathsf{corr}}]^{\mathsf{cov}}$ sends the message (cheat, i) to S_{corr} (simulating $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$) the simulator S_{corr} forwards it to $\mathcal{F}_{\mathsf{on}}$.
   (a) If S_{corr} receives back $\mathsf{undetected}$ and the inputs of the honest parties $\overrightarrow{I}$ from $\mathcal{F}_{\mathsf{on}}$ our simulator behaves as follow: 1) she sends $\mathsf{undetected}$ to $[S_{\mathsf{corr}}]^{\mathsf{cov}}$ receiving the correlated randomesses $\overrightarrow{R}$ of the honest parties; 2) S_{corr} engages an execution of Π^{on} with $\mathcal{A}$ acting as the honest parties using $\overrightarrow{I}$ and $\overrightarrow{R}$. In the end of the execution S_{corr} gets some output for the honest parties that is forwarded to $\mathcal{F}_{\mathsf{on}}$.
   (b) If S_{corr} receives back $\mathsf{detected}$ form $\mathcal{F}_{\mathsf{on}}$ our simulator forwards it to $[S_{\mathsf{corr}}]^{\mathsf{cov}}$.
3. If S_{corr} sends the inputs of the malicious party $[S_{\mathsf{corr}}]^{\mathsf{cov}}$ stores them for the online phase.

If no cheating happened S_{corr} runs S_{on} during the execution of Π^{on} with $\mathcal{A}$. Similarly to before when S_{on} invokes $\mathcal{F}_{\mathsf{on}}$ our simulator S_{corr} acts as a proxy between her and $\mathcal{F}_{\mathsf{on}}$, in particular:

- If S_{on} sends the inputs of the adversaries, S_{corr} forwards to Π^{on} the output of the adversary received by $\mathcal{F}_{\mathsf{on}}$.
- If S_{on} sends (abort, i), S_{corr} forwards $(\mathsf{corrupted}, i)$ to $\mathcal{F}_{\mathsf{on}}$.

Observe that if cheating was detected in the preprocessing, then the simulated view is indistinguishable from a real one by assumption on the security of $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$. If no cheating happened, then the actively secure and covertly secure versions of $\mathcal{F}_{\mathsf{on}}$ have identical input/output behaviours and thus the simulated views are also indistinguishable by assumption. □

Lemma 22. *Suppose Π^{on} securely realizes $\mathcal{F}_{\mathsf{on}}$ with active security (and abort) in the $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$-hybrid model, and that $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$ securely realizes $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ with covert security and public verifiability. Then, the combined protocol $(\Pi^{\mathsf{on}}, [\Pi_{\mathsf{corr}}]^{\mathsf{cov}})$ securely realizes $\mathcal{F}_{\mathsf{on}}$ with covert security and public verifiability.*

Proof. The proof of covert security follows similarly to the one for lemma 21 (the only difference is that the simulator obtains/forwards abort to the functionality $\mathcal{F}_{\mathsf{on}}$). Regarding the public verifiability, we observe that Π^{on} is active secure, which implies that the adversary $\mathcal{A}$ could only cause an abort in this part of the protocol. Therefore, the public verifiability of $(\Pi^{\mathsf{on}}, [\Pi_{\mathsf{corr}}]^{\mathsf{cov}})$ is implied by the public verifiability of $[\Pi_{\mathsf{corr}}]^{\mathsf{cov}}$. □

E Further Details on Efficiency Analysis

In Table 2, we present further concrete numbers comparing the costs of different passive, covert and active protocols (without identifiable abort), and with varying choices of the covert replication factor k (which gives deterrence $1-1/k$).

F Proof of Theorem 1

Proof. We first prove that Π^{idc} has passive security with consistent identifiability. It is straightforward that Π^{idc} securely computes $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ with passive security, from the simple fact that Π does, and the only additional messages sent in a passive execution are digital signatures on known messages. These are easily simulated, since the simulator can choose secret signing keys on behalf of the honest parties.

We now consider the identifiability property (Definition 4). We need to show that whenever an honest party outputs abort_i, all honest parties unanimously output abort_i for the same index $i \in A$. Note that if some honest party aborts, the parties always output the same index i, since the criteria for aborting in step 2a is based on public verification of a signature which was broadcast by some party P_i. Furthermore, by the correctness of the signature scheme, P_i must always be corrupt, since if P_i were honest then the signature would verify.

For the identifiable cheating property, we consider an adversary $\mathcal{A}$ who succeeds in experiment $\mathsf{Exp}^{\mathsf{idc}}_{\mathcal{A},\Pi}(\lambda)$ with some noticeable probability. If $\mathcal{A}$ wins according to condition 7a in the experiment, then the execution of Π was dishonest, yet no party was identified by the $\mathsf{Identify}$ procedure. By Definition 5, there must exist an honest party P_j, corrupt party P_i and round ρ where P_i's message $m^{\rho}_{i,j}$ was generated incorrectly, that is, $m^{\rho}_{i,j} \neq \tilde{m}^{\rho}_{i,j}$. Furthermore, if $\mathsf{Identify}$ outputs $\perp$ then the honest P_j's $\mathsf{Certify}$ algorithm did not produce a certificate. This means we must have $\mathsf{H}(m^{\rho}_{i,j}) = \mathsf{H}(\tilde{m}^{\rho}_{i,j})$, therefore $\mathcal{A}$ has found a collision in H.

On the other hand, if $\mathcal{A}$ wins according to condition 7b, we argue that $\mathcal{A}$ must have forged a signature under some honest party's secret key. Let i' be the index of the honest party who was identified, and cert_i the certificate with the smallest ρ_i which shows that $P_{i'}$ cheated. Since we choose the smallest ρ_i, it holds that in all previous rounds every message received by $P_{i'}$ was generated correctly. Therefore, the round ρ_i messages from $P_{i'}$ were also generated correctly, and would contain signatures on $(\rho_i \| P_{i'} \| P_i \| \mathsf{H}(\tilde{m}^{\rho_i}_{i',i}))$. To identify $P_{i'}$ as a cheater requires a signature on a different hash value with the same prefix, but since the round numbers are unique, this can only be done by forging a new signature. Therefore, $\mathcal{A}$ breaks the EUF-CMA security of the signature scheme. □

G Proof of Theorem 3

Proof. We first prove that $\Pi^{\mathsf{w\text{-}idc}}$ has passive security with consistent identifiability. It is straightforward that $\Pi^{\mathsf{w\text{-}idc}}$ securely computes $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ with passive security, from the simple fact that Π does, and the only additional messages sent in a passive execution are digital signatures on known messages. These are

easily simulated, since the simulator can choose secret signing keys on behalf of the honest parties.

We now argue that $\Pi^{\mathsf{w\text{-}idc}}$ supports publicly verifiable cheating from random tapes (in the rest of the proof, we will use the sentence "valid certificate" to indicate a certificate accepted by $\mathsf{Sentence}$):

Defamation-freeness. Suppose by contradiction that the $\mathcal{A}$ succeeds with non-negligible probability in step 8b of $\mathtt{Exp}^{\mathsf{pvc}}_{\mathcal{A},\Pi}(\lambda, x)$. Wlog let us assume the $\mathcal{A}$ is able to produce a valid certificate $\mathsf{cert}^* = (\sigma_{\mathsf{Agg}}, \sigma^*, h^*, i^*, j, \rho^*)$ that accuses honest party P_{i^*}. If $\mathsf{Sentence}$ in step 5 outputs vk_{i^*}, then the adversary managed to output a valid signature σ^* w.r.t. vk_{i^*} of an hash that does not correspond to the honest message $\widetilde{m}^{\rho^*}_{i^*,j}$. In this case since P_{i^*} is honest we can give a reduction to the unforgeability of the signature scheme. Note that we are guaranteed that the message $\widetilde{m}^{\rho^*}_{i^*,j}$ (computed by $\mathsf{Sentence}$) is the same as the message sent by the honest P_{i^*} during the protocol, because of the valid signature on the previous message hashes $\mathcal{M}$, and the unforgeability of the signature scheme and the collision resistance of H.

Public verifiability. Suppose by contradiction that the $\mathcal{A}$ succeeds with non-negligible probability in step 8a of $\mathtt{Exp}^{\mathsf{pvc}}_{\mathcal{A},\Pi}(\lambda, x)$. This implies that wlog a corrupted P_{i^*} in some round ρ^* sent for the first time a message $m^{\rho^*}_{i^*,j}$ to an honest P_j that is inconsistent w.r.t. his random tapes and NMF. We note that $\mathcal{A}$ succeeds in $\mathtt{Exp}^{\mathsf{pvc}}_{\mathcal{A},\Pi}(\lambda, x)$ only when the execution of Π did not abort. Therefore in round ρ^* P_j has collected (1) $\{m^{\rho}_{j,i}, \sigma^{\rho}_{j,i}\}_{j \neq i, \rho \in [\rho']}$ where $\rho' < \rho^*$; (2) a signature from P_{i^*} on $m^{\rho^*}_{i^*,j}$. Looking at the Accuse algorithm we can conclude that (1) and (2) are sufficient to produce a valid certificate against P_{i^*}.

□

H Proof of Theorem 4

Proof. Let S_{corr} be the simulator for Π_{corr}. Let $\mathcal{A}$ be the adversary who corrupts a subset $\mathbb{A} \subset \{P_1, \ldots, P_n\}$ of parties, where $|\mathbb{A}| \leq n-1$. Let $\mathbb{P}$ be the set of honest parties, i.e. $\mathbb{P} = \{P_1, \ldots, P_n\} \setminus \mathbb{A}$.

We are now ready to describe our simulator S:

1. Generate signature key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ for each party $P_i \in \mathbb{P}$ and send pk_i to $\mathcal{A}$. For each $P_i \in \mathbb{A}$, the adversary outputs a public key pk_i.
2. For each $i \in [n]$ for each $P_j \in \mathbb{A}$, the adversary outputs $(\mathsf{comFlip}, (i,1)), \ldots, (\mathsf{comFlip}, (i,k))$. For each $i \in [n]$, the simulator picks uniformly random values $r_{i,1}, \ldots, r_{i,k}$ and sends back $(\mathsf{com}, (i,1)), \ldots, (\mathsf{com}, (i,k))$ to $\mathcal{A}$.
3. For each $i \in [n]$ for each $P_j \in \mathbb{A}$, the adversary sends $(\mathsf{openOne}, (i,1), i), \ldots, (\mathsf{openOne}, (i,k), i)$.
4. For each $P_i \in \mathbb{A}$, the simulator sends $(\mathsf{openOne}, r_{i,1}), \ldots, (\mathsf{openOne}, r_{i,k})$ to $\mathcal{A}$.
5. All parties jointly execute Π_{corr} in parallel k times, where party P_i uses random tape $r_{i,j}$ in the j-th execution of the protocol where S acts on behalf

of parties $P_i \in \mathbb{P}$ and $\mathcal{A}$ acts on behalf of parties $P_i \in \mathbb{A}$. Let $R_{i,j}$ be the output of party P_i in execution j.

6. For each $P_j \in \mathbb{A}$, the adversary outputs $(\mathsf{coinFlip}, 0)$.
7. The simulator S checks whether the adversary misbehaved in Step 5. Let X be the set of executions of Π_{corr} where $\mathcal{A}$ cheated.
   (a) If $|X| \geq 2$, then the simulator picks a uniformly random $j^* \in [k]$ and sets $\mathsf{flag} = \mathsf{blatant\_cheat}$.
   (b) If $|X| = 1$, then let $\widehat{j}$ be the execution of Π^{corr} in which $\mathcal{A}$ cheated. Let $\widehat{i}$ be the smallest index in the set of parties that cheated in execution $\widehat{j}$. The simulator sends $(\mathsf{cheat}, \widehat{i})$ to the ideal functionality $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$.
      i. If $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ sends $(\mathsf{corrupted}, \widehat{i})$, the simulator picks a uniformly random value $j^* \in [k]$ s.t. $j^* \neq \widehat{j}$ and sets $\mathsf{flag} = \mathsf{detected}$.
      ii. If $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ sends $(\mathsf{undetected})$ the simulator sets $j^* = \widehat{j}$ and $\mathsf{flag} = \mathsf{undetected}$.
   (c) If $|X| = 0$, then set $\mathsf{flag} = \mathsf{allgood}$. The simulator picks uniformly random value j^*.
8. S sends $(\mathsf{coinFlip}, 0, j^*)$ to $\mathcal{A}$.
9. For $j \in [k] \backslash \{j^*\}$ for each $i \in [n]$ S sends $(\mathsf{openAll}, r_{i,j})$ to $\mathcal{A}$ and the adversary sends $(\mathsf{openAll}, i, j)$ to S.
10. For $j \in [k] \setminus \{j^*\}$ for each party $P_i \in \mathbb{P}$, the simulator computes $\mathsf{cert}_{i,j} \leftarrow \mathsf{Certify}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, \boldsymbol{r}_j, \mathsf{view}_{i,j})$, where $\mathsf{view}_{i,j}$ is P_i's view in the j-th execution of Π_{corr} and sends $\mathsf{cert}_{i,j}$ to $\mathcal{A}$.
11. $\mathcal{A}$ sends $\mathsf{cert}^*_{i,j}$ for $j \in [k] \setminus \{j^*\}$ for each party $P_i \in \mathbb{A}$.
12. Depending on the flag do:
   (a) If $\mathsf{flag} = \mathsf{blatant\_cheat}$ or $\mathsf{flag} = \mathsf{detected}$, then the simulator broadcasts $(\mathsf{corrupted}, \widehat{i})$ to $\mathcal{A}$ and the ideal functionality. The simulator outputs whatever $\mathcal{A}$ outputs and terminates.
   (b) $\mathsf{flag} = \mathsf{undetected}$, then for each party $P_i \in \mathbb{P}$, the simulator S sends $R_{(i,\widehat{j})}$ as the output of P_i to the ideal functionality. The simulator outputs whatever $\mathcal{A}$ outputs and terminates.
   (c) if $\mathsf{flag} = \mathsf{allgood}$ S uses $\{r_{i,j^*}\}_{P_i \in \mathbb{A}}$ to extract adversary's input $R_{\mathbb{A}} = \{R_i\}_{P_i \in \mathbb{A}}$ from the j^*-th execution of Π_{corr}[9] sends $R_{\mathbb{A}}$ to $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$. S rewinds $\mathcal{A}$ back to Step 4, running part B, until it terminates. The simulator outputs whatever $\mathcal{A}$ outputs and terminates.

We remark here, to not overburden the description of our simulator S, that in case $\mathcal{A}$ stops responding while executing a functionalities (internally emulated by the simulator), the functionalities should send to the honest parties abort_j, where j is the index of the parties that stops responding. In this case, instead, S sends $(\mathsf{corrupted}, j)$ to $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$.

We will now proceeds through a series oh hybrid experiments in oder to prove that the joint distribution of the view of $\mathcal{A}$ and the output of the honest parties in the ideal execution is computationally indistinguishable from the

[9] Note that since it holds that $\mathsf{flag} = \mathsf{allgood}$, then Π_{corr} was executed honestly by $\mathcal{A}$, therefore $R_{\mathbb{A}}$ are taken from the distribution of valid inputs, i.e. inputs for which is possible for $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ to reverse sample the correlated randomness for the honest parties.

joint distribution of the view of $\mathcal{A}$ and the output of honest parties in a real protocol execution. The hybrid experiments are listed below. The output of the experiments is defined as the view of $\mathcal{A}$ and the output of the honest parties.

- Exp_0 : In this game, the simulator S_0 acts as the simulator S described above, moreover S_0 has access to the internal state of $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$, therefore S_0 chooses the output values of the honest parties. Note that towards the honest parties S_0 is acting as $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ does without any modifications therefore the output of Exp_0 and the output of ideal world experiment are identically distributed.
- In Exp_1 : The experiment Exp_0 is modified in the following way: the simulator S_1 choses at random an index $\bar{j} \in [k]$ at the beginning of the Exp_1, then in Step 12 S_1 acts as follows:
  1. If $|X| \geq 2$, then the simulator picks a uniformly random $j^* \in [k]$ and sets $\mathsf{flag} = \mathsf{blatant\_cheat}$.
  2. If $|X| = 1$, if $\bar{j} \in X$ sets $\mathsf{flag} = \mathsf{undetected}$ and $j^* = \bar{j}$; otherwise sets $\mathsf{flag} = \mathsf{detected}$ and picks a uniformly random value $j^* \in [k]$ s.t. $j^* \neq \widehat{j}$, where $\widehat{j}$ is the execution of Π_{corr} in which $\mathcal{A}$ cheated.
  3. If $|X| = 0$, then set $\mathsf{flag} = \mathsf{allgood}$. The simulator sets $j^* = \bar{j}$.

  The output of the experiments Exp_1 and Exp_0 are identically distribute since when $|X| = 1$ with probability ϵ it happens that $\bar{j} \notin X$.
- In Exp_2 : The experiment Exp_1 is modified in the following way. The simulator S_2, in case of $\mathsf{flag} = \mathsf{allgood}$ outputs as the output of the honest parties the output of j^*-th execution of Π_{corr}, instead of relying on the output of $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ (which is internally emulated by the simulator after Exp_0).
  The output of the experiments Exp_1 and Exp_2 are computationally indistinguishable due to the semi-honest security of Π_{corr}.
- Exp_3: We modify the experiment by letting S_3 in step 12a behave identically to step 8 in the real protocol. The simulator S_3, acting as the honest parties, receives certificates $(\mathsf{cert}_{(1,j)}, \ldots, \mathsf{cert}_{(n,j)})$ for each execution $j \neq j^*$ and computes $v_j \leftarrow \mathsf{Ver}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n, \boldsymbol{r}_j, \mathsf{cert}_{(1,j)}, \ldots, \mathsf{cert}_{(n,j)})$. Let J be the set of indices with $v_j \neq \perp$. If $J \neq \emptyset$, then S_3 acting as $P_i \in \mathbb{P}$ broadcasts $(\mathsf{corrupted}, v_j)$ with the smallest j from J. Note that $\mathsf{Exp}_{2'}$ and Exp_3 only differ when $\mathcal{A}$ misbehaves in one or more executions, which are also opened, and manages to produce partial certificates that do not lead to a corrupted party being accused by $\mathsf{Identify}$. The indistinguishability of the two experiments thus reduces to the IDC security property of Π_{corr}.

The proof ends observing that in Exp_3 the simulator S_3 does not need anymore to have access to the internal state of the functionalities (i.e. there is no needing to simulate the functionalities w.r.t. the adversary) therefore Exp_3 and the real world experiment are identically distributed. □

I Proof of Theorem 5

Proof. We will now arguing separately that $[\Pi_{\mathsf{corr}}]^{\mathsf{pvcov}}$ enjoys public verifiability, covert security with ϵ-deterrence factor and defamation-freeness.

We start by showing that $[\Pi_{\mathsf{corr}}]^{\mathsf{pvcov}}$ enjoys covert security with ϵ-deterrence factor.

Covert security. *The simulator.* Let S_{corr}, S_{flip}, S_{check} be the simulator respectively for Π_{corr}, Π_{flip}, Π_{check}. Let $\mathcal{A}$ be the adversary who corrupts a subset $\mathbb{A} \subset \{P_1, \ldots, P_n\}$ of parties, where $|\mathbb{A}| \leq n-1$. Let $\mathbb{P}$ be the set of honest parties, i.e. $\mathbb{P} = \{P_1, \ldots, P_n\} \setminus \mathbb{A}$. Our simulator S works as follows:

Simulation part A:

1. Generate signature key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ for each party $P_i \in \mathbb{P}$ and send pk_i to $\mathcal{A}$. For each $P_i \in \mathbb{A}$, the adversary outputs a public key pk_i.
2. S activates S_{flip} and acts as a proxy between $\mathcal{A}$ and S_{flip} in the execution of Π_{flip}. Moreover S simulates $\mathcal{F}_{\mathsf{FLIP}}$ to S_{flip} sending for $P_i \in \mathbb{A}$ (in case S_{flip} does not abort the computation)

$$\left(\mathsf{com}_{i,1}, \ldots, \mathsf{com}_{i,k}\right) \text{ and } \left(\mathsf{dec}_{i,1}, \ldots, \mathsf{dec}_{i,k}\right),$$

   In the end of this phase $\mathcal{A}$ for $P_i \in \mathbb{A}$ obtains:

$$C^{\mathsf{rnd}} = \begin{pmatrix} \mathsf{com}_{1,1}, \ldots, \mathsf{com}_{1,k} \\ \vdots \\ \mathsf{com}_{n,1}, \ldots, \mathsf{com}_{n,k} \end{pmatrix}$$

   The simulator S runs S_{flip} with the command $\mathsf{openAll}$ in order to get the decommitments for the honest parties' random tapes $\{r_{i,j}\}_{P_i \in \mathbb{P}, j \in [k]}$, which are chosen uniformly at random.
3. All parties jointly execute Π_{corr} in parallel k times, where party P_i uses random tape $r_{i,j}$ w.r.t $.\mathsf{com}_{i,j}$ in in the j-th execution of the protocol where S acts on behalf of parties $P_i \in \mathbb{P}$ and $\mathcal{A}$ acts on behalf of parties $P_i \in \mathbb{A}$. Let $R_{i,j}$ be the output of party P_i in execution j. Let $\mathsf{view}_{i,j}$ be the view of P_i in the j-th execution of Π_{corr}. After the executions, for $j \in [k]$, on behalf of each $P_i \in \mathbb{P}$, S runs $\mathsf{cert}_{i,j} \leftarrow \mathsf{GatherEvidence}(\mathsf{view}_{i,j})$.
4. For each party $P_i \in \mathbb{P}$, S picks a uniformly random encryption keys $K_{i,1}, \ldots, K_{i,k}$ and broadcasts encryptions $c_{i,j} \leftarrow \mathsf{sEnc}_{k_{i,j}}(\mathsf{dec}_{i,j}, \mathsf{cert}_{i,j})$ for $j \in [k]$. For each party $P_i \in \mathbb{A}$ for $j \in [k]$, S receives $c_{i,j}$ from $\mathcal{A}$. Let CT be the set of all $n \times k$ ciphertexts.
5. The simulator picks uniformly random permutation γ.
6. S activates S_{check} and acts as a proxy between $\mathcal{A}$ and S_{check} in the execution of Π_{check}. S obtains $(K_{i,1}, \ldots, K_{i,k})$ for $i \in \mathbb{A}$ from S_{check}. Moreover (in case S_{check} does not abort the computation) S simulates $\mathcal{F}_{\mathsf{check}}$ to S_{check} sending s_i for $P_i \in \mathbb{A}$, which is the i-th share of

$$\boldsymbol{s} = \left(\begin{pmatrix} K_{1,\gamma(1)}, \ldots, K_{n,\gamma(1)} \\ \vdots \\ K_{1,\gamma(k-1)}, \ldots, K_{n,\gamma(k-1)} \end{pmatrix}, \gamma \right)$$

7. For each party $P_i \in \mathbb{P}$, S computes $\mathsf{dec}_i^{\mathsf{share}}$ and $\mathsf{com}_i^{\mathsf{share}}$ by using Π_{com} to commit to s_i and broadcasts $\mathsf{com}_i^{\mathsf{share}}$. For each party $P_i \in \mathbb{A}$, S receives $\mathsf{com}_i^{\mathsf{share}}$ from $\mathcal{A}$. Let $C^{\mathsf{share}} = (\mathsf{com}_1^{\mathsf{share}}, \ldots, \mathsf{com}_n^{\mathsf{share}})$.

8. For each party $P_i \in \mathbb{P}$, S generates a puzzle $\mathsf{puz}_i \leftarrow \mathsf{pGen}\left(t, \mathsf{dec}_i^{\mathsf{share}}\right)$, where $t = 5$, commits to puz_i using Π_{com} obtaining $\mathsf{com}_i^{\mathsf{puz}}, \mathsf{dec}_i^{\mathsf{puz}}$, and broadcasts $\mathsf{com}_i^{\mathsf{puz}}$. For each party $P_i \in \mathbb{A}$, S receives puz_i from $\mathcal{A}$. Let $\overrightarrow{\mathsf{puz}}$ be the vector of all puzzles $\mathsf{puz}_1, \ldots, \mathsf{puz}_n$.
9. If in one of the previous steps $\mathcal{A}$ stops responding S sends abort to $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$.
10. After receiving $\mathsf{com}_j^{\mathsf{puz}}$ for all $j \neq i$, each P_i broadcasts $\mathsf{dec}_i^{\mathsf{puz}}$. Each party checks whether all the puzzle commitments it receives are valid decommitments and aborts if this is not the case.
11. For each party $P_i \in \mathbb{P}$, S computes signature $\sigma_i \leftarrow \mathsf{Sig}_{\mathsf{sk}_i}\left(C^{\mathsf{rnd}}, C^{\mathsf{share}}, CT, \overrightarrow{\mathsf{puz}}\right)$ and broadcasts it to every other party. If S obtains an invalid signature or no signature at all (in this round) from $\mathcal{A}$, then S sends abort to $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$.
12. For each party $P_i \in \mathbb{P}$, S broadcasts $\mathsf{dec}_i^{\mathsf{share}}$. For each party $P_i \in \mathbb{A}$, S receives $\mathsf{dec}_i^{\mathsf{share}}$ from $\mathcal{A}$. Let $J \subset [n]$ be the set indices belonging to parties that did not broadcast this message. If $J \neq \emptyset$, the each party P_i solves all puzzles puz_j with $j \in J$ and if one or more puzzles are not solvable in time t, then S does the following :
    - Send $(\mathsf{corrupted}, j^*)$ to $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$, where j^* is the smallest index belonging to a not solvable puzzle;
    - Compute the corresponding certificate as an honest party would do (note that all the information are available from the previous simulation steps) and send it to $\mathcal{A}$;
    - Terminate giving in output the output of the adversary.

    If all relevant puzzles are solvable, then replace missing decommitments with the ones from the solved puzzles.
13. Let S be the set of invalid decommitments. Solve corresponding puzzles to see whether they contain the valid decommitment. If yes, then continue as below using the valid decommitment, otherwise S does the following:
    - Send $(\mathsf{corrupted}, j^*)$ to $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$, where j^* is the smallest index belonging to a not valid decommitment;
    - Compute the corresponding certificate as an honest party would do (note that all the information are available from the previous simulation steps) and send it to $\mathcal{A}$;
    - Terminate giving in output the output of the adversary.
14. If any of the decommitted shares is invalid, then S sends abort to $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$.
15. Each party uses the shares to reconstruct $K_{1,\gamma(j)}, \ldots, K_{n,\gamma(j)}$ for $j \in [k-1]$.
16. S checks whether any of the decryptions fail and if it does then we set the corresponding plaintext to be $\perp$.
17. S checks whether the reconstructed random tapes and decommitments for each execution $j \in [k-1]$ match the corresponding commitments in matrix C^{rnd}. If not the simulator does the following:
    - Send $(\mathsf{corrupted}, i^*)$ to $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$, where j^* is the smallest execution index with a mismatch and i^* is the smallest index of a party within that execution that produces a mismatch.
    - Compute the corresponding certificate as an honest party would do (note that all the information are available from the previous simulation steps) and send it to $\mathcal{A}$;

– Terminate giving in output the output of the adversary.

18. The simulator S checks whether the adversary misbehaved in Step 3. Let X be the set of executions of Π_{corr} where $\mathcal{A}$ cheated.
    (a) If $|X| \geq 2$, then a blatant cheat and the simulator already detected this event in previous step.
    (b) If $|X| = 1$, then let $\widehat{j}$ be the execution of Π^{corr} in which $\mathcal{A}$ cheated. Let $\widehat{i}$ be the smallest index in the set of parties that cheated in execution j^*. The simulator sends $(\mathsf{cheat}, \widehat{i})$ to the ideal functionality $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$.
        i. If $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ sends $(\mathsf{corrupted}, \widehat{i})$ sets $\mathsf{flag} = \mathsf{detected}$.
        ii. If $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ sends $(\mathsf{undetected})$ the simulator $\mathsf{flag} = \mathsf{undetected}$.
    (c) If $|X| = 0$, then set $\mathsf{flag} = \mathsf{allgood}$.
19. Depending on the flag do:
    (a) If $\mathsf{flag} = \mathsf{detected}$ then the simulator broadcasts $(\mathsf{corrupted}, \widehat{i})$ to $\mathcal{A}$ and the ideal functionality.
    If $\gamma(k) \neq \widehat{j}$:
        – S computes the corresponding certificate as an honest party would do (note that all the information are available from the previous simulation steps) and send it to $\mathcal{A}$; S outputs whatever $\mathcal{A}$ outputs and terminates.
    Otherwise:
        – S rewinds $\mathcal{A}$ back to step 5 (computing all the other steps as before). S keeps on rewinding (following the above strategy) until the simulation reaches step 18, at that point S omputes the corresponding certificate as an honest party would do (note that all the information are available from the previous simulation steps) and send it to $\mathcal{A}$; S outputs whatever $\mathcal{A}$ outputs and terminates.
    (b) if $\mathsf{flag} = \mathsf{undetected}$ then for each party $P_i \in \mathbb{P}$, the simulator S sends $R_{(i,\widehat{j})}$ as the output of P_i to the ideal functionality.
    If $\gamma(k) = \widehat{j}$ then S outputs whatever $\mathcal{A}$ outputs and terminate, otherwise: S rewinds $\mathcal{A}$ back to step 5 (computing all the other steps as before). S keeps on rewinding (following the above strategy) until the simulation reaches step 18, at that point S outputs whatever $\mathcal{A}$ outputs and terminate.
    (c) if $\mathsf{flag} = \mathsf{allgood}$ S rewinds $\mathcal{A}$ back to Step 3, running part B, until it terminates[10].

Simulator part B:

1. The simulator picks uniformly random value j^*.
2. For every $i \in \mathbb{P}$, for execution j^*, the simulator sets $\mathsf{com}_{i,j^*}, \mathsf{dec}_{i,j^*}$ s.t. the message committed in com_{i,j^*} is 0. The remaining random tapes remain uniformly random as before in part A. S activates S_{flip} and acts as a proxy between $\mathcal{A}$ and S_{flip} in the execution of Π_{flip}. Moreover S simulates $\mathcal{F}_{\mathsf{FLIP}}$ to S_{flip} sending for $P_i \in \mathbb{A}$ (in case S_{flip} does not abort the computation).

[10] Our simulation can be made to run in expected polynomial time via Goldreich and Kahan [21].

$$\left(\mathsf{com}_{i,1},\ldots,\mathsf{com}_{i,k}\right) \text{ and } \left(\mathsf{dec}_{i,1},\ldots,\mathsf{dec}_{i,k}\right),$$

In the end of this phase $\mathcal{A}$ for $P_i \in \mathbb{A}$ obtains:

$$C^{\mathsf{rnd}} = \begin{pmatrix} \mathsf{com}_{1,1},\ldots,\mathsf{com}_{1,k} \\ \vdots \\ \mathsf{com}_{n,1},\ldots,\mathsf{com}_{n,k} \end{pmatrix}$$

3. All parties jointly execute Π_{corr} in parallel k times:
   In all executions the simulator S honestly acts on behalf of the parties in $\mathbb{P}$ as before. After the executions, for $j \in \{1,\ldots,k\}/\{j^*\}$, on behalf of each $P_i \in \mathbb{P}$, S runs $\mathsf{cert}_{i,j} \leftarrow \mathsf{GatherEvidence}(\mathsf{view}_{i,j})$. In execution j^* of Π_{corr} the simulator will use a fresh random tape and not the one committed in C^{rnd}.
4. For each party $P_i \in \mathbb{P}$, S picks a uniformly random encryption keys $K_{i,1},\ldots,K_{i,k}$. S broadcasts encryptions $c_{i,j} \leftarrow \mathsf{sEnc}_{k_{i,j}}(\mathsf{dec}_{i,j}, \mathsf{cert}_{i,j})$ for $j \in \{1,\ldots,k\}/\{j^*\}$ and S broadcasts encryptions $c_{i,j^*} \leftarrow \mathsf{sEnc}_{k_{i,j^*}}(0^l)$. For each party $P_i \in \mathbb{A}$ for $j \in [k]$, S receives $c_{i,j}$ from $\mathcal{A}$. Let CT be the set of all $n \times k$ ciphertexts.
5. The simulator picks uniformly random permutation γ over $[k]$ such that $\gamma(k) = j^*$.

For the remaining steps simulator in Part B is acting exactly as the simulator in part A until step 3 with the differences: that if $\mathcal{A}$ aborts or misbehaves, it rewinds back to the start of part B. When the simulation reach step 18, S uses $\{r_{i,j^*}\}_{P_i \in \mathbb{A}}$ to extract adversary's input $R_{\mathbb{A}} = \{R_i\}_{P_i \in \mathbb{A}}$[11] from the j^*-th execution of Π_{corr} and sends $R_{\mathbb{A}}$ to $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$, finally S outputs whatever $\mathcal{A}$ outputs and terminate.

We will now proceeds through a series oh hybrid experiments in oder to prove that the joint distribution of the view of $\mathcal{A}$ and the output of the honest parties in the ideal execution is computationally indistinguishable from the joint distribution of the view of $\mathcal{A}$ and the output of honest parties in a real protocol execution. The hybrid experiments are listed below. The output of the experiments is defined as the view of $\mathcal{A}$ and the output of the honest parties.

- Exp_0 : In this game, the simulator S_0 acts as the simulator S described above, moreover S_0 has access to the internal state of $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$, therefore S_0 chooses the output values of the honest parties. Note that towards the honest parties S_0 is acting as $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ does without any modifications therefore the output of Exp_0 and the output of ideal world experiment are identically distributed.
- In Exp_1 : The experiment Exp_0 is modified in the following way: the simulator S_1 choses at random an index $\bar{j} \in [k]$ and in step 18 in part A S_1 acts as follows:
  1. If $|X| = 1$, if $\bar{j} \in X$ sets $\mathsf{flag} = \mathsf{undetected}$ otherwise sets $\mathsf{flag} = \mathsf{detected}$.
  2. If $|X| = 0$, then set $\mathsf{flag} = \mathsf{allgood}$. The simulator sets $j^* = \bar{j}$.

[11] Note that since it holds that $\mathsf{flag} = \mathsf{allgood}$, then Π_{corr} was executed honestly by $\mathcal{A}$, therefore $R_{\mathbb{A}}$ are taken from the distribution of valid inputs, i.e. inputs for which is possible for $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ to reverse sample the correlated randomness for the honest parties.

The output of the experiments Exp_1 and Exp_0 are identically distribute since when $|X| = 1$ with probability ϵ it happens that $\bar{j} \notin X$.

- In Exp_2 : The experiment Exp_1 is modified in the following way. The simulator S_2, in case of $\mathsf{flag} = \mathsf{allgood}$ outputs as the output of the honest parties the output of $\bar{j}$-th execution of Π_{corr}, instead of relying on the output of $\mathcal{F}_{\mathsf{corr}}^{\mathcal{D}}$ (which is internally emulated by the simulator after Exp_0).
  The output of the experiments Exp_1 and Exp_2 are computationally indistinguishable due to the security of Π_{corr}.
- In Exp_3 : The experiment Exp_2 is modified in the following way. In step 3 of part B S_3 according to the part A of the simulation (i.e., the random tapes committed for the $\bar{j}$-th execution of Π_{corr} are effectively the one used for executing Π_{corr}). The output of the experiments Exp_3 and Exp_2 are computationally indistinguishable due to the security of Π_{flip}.
- In Exp_4 : The experiment Exp_3 is modified in the following way: The ciphertexts CT are computes as in Part A of the simulation. The output of the experiments Exp_3 and Exp_4 are computationally indistinguishable due to the CPA security of $(\mathsf{sEnc}, \mathsf{sDec})$.
- In Exp_5 : The experiment Exp_4 is modified in the following way: Π_{check} is executing as in part A of the simulation, i.e., on input a uniformly random permutation γ. The output of the experiments Exp_5 and Exp_4 are computationally indistinguishable due to the security of Π_{check}.
- In $\mathsf{Exp}_{5'}$: We observe that in case of no cheating, we do the same thing in Part A and B. Thus, we can 'collapse' the two parts of the simulator given that in both parts everithing correlated to the $\bar{j}$-th execution of Π_{corr} is executed using the procedure of the honest parties. Therefore the output of the experiments $\mathsf{Exp}_{5'}$ and Exp_5 are statistically close[12].
- Exp_6: We modify the experiment by letting S_6 in step 18 if $\mathsf{flag} = \mathsf{detected}$ behave identically to step 14 in the real protocol. Note that $\mathsf{Exp}_{5'}$ and Exp_6 only differ when $\mathcal{A}$ misbehaves in one or more executions, which are also opened, and manages to produce partial certificates that do not lead to a corrupted party being accused by $\mathsf{Sentence}$. Thus, we rely on the public verifiability cheating from random tapes of Π_{corr} to claim the indistinguishability of the two experiments.
- In Exp_7 : The experiment Exp_6 is modified in the following way: Π_{check} is executing as in honestly. The output of the experiments Exp_7 and Exp_6 are computationally indistinguishable due to the security of Π_{check}.
- In Exp_8 : The experiment Exp_8 is modified in the following way: Π_{flip} is executing as in honestly. The output of the experiments Exp_7 and Exp_8 are computationally indistinguishable due to the security of Π_{flip}.

The proof ends observing that in Exp_8 the simulator S_8 does not need anymore to have access to the internal state of the functionalities (i.e. there is no needing to simulate the functionalities w.r.t. the adversary) therefore Exp_6 and the real world experiment are identically distributed.

[12] The statistical difference comes from a differing number in rewinding steps, which can influence the abort probability by a statistically negligible amount.

Defamation-Freeness. In $[\Pi_{\mathsf{corr}}]^{\mathsf{pvcov}}$ is possible to output a certificate with respect to four types of different cheating attempts. For each case, we will now argue that an adversary $\mathcal{A}$ is not able to output a valid certificate, which accuses an honest party. Suppose by contradiction that this is not the case, i.e. there exists a non-negligible probability that $\mathcal{A}$ is able to output an accepting certificate $\mathsf{cert} = (c, \mathsf{aux})$, then we can distinguishes 4 cases:

- $\mathbf{c \in \{1, 2, 3\}}$:
  Let us assume wlog that cert is accusing the honest party P_i. The important two observations are 1) since P_i is honest she follows the protocol honestly and therefore she is producing a solvable puzzle (resp. valid decommitments of C^{rnd}, valid decommitments of C^{share}); 2) if $\mathcal{A}$ outputs a valid cert, then it contains a valid signature of P_i. From these two observations we can conclude that the adversary would need to break unforgeability of the signature scheme.
- $\mathbf{c = 4}$:
  If $\mathcal{A}$ outputs a valid cert with non negligible probability then it is possible to show a reduction to the public verifiability cheating from random tapes of Π_{corr} (in particular to the defamation-freeness property of Π_{corr}).

Public Verifiability. Suppose by contradiction that wlog party P_{i^*} misbehaves in step 3 and the honest parties are not able to produce a valid certificate against P_{i^*}. Let j^* be the smallest index among the checked execution of Π_{corr} where P_{i^*} cheats. If this is the case, then we would like to show a reduction to the public verifiability cheating from random tapes of the j^*-th execution of Π_{corr}. In order to build this reductions we need to argue that 1) in C^{rnd} are committed the output of Π_{flip} 2) the decommitments of C^{rnd} and the partial certificate are retrievable from $\overrightarrow{\mathsf{puz}}$. We observe that (1) follows from the UC-security of Π_{flip}; We now argue using a case analysis that also (2) is true. In more details we want to prove the following invariant $\mathcal{I}$: if P_{i^*} cheats in any subsequents steps after step 3 one of this three events must happen: the protocol aborts (independently of the cheating attempts); the honest parties are able to produce a valid certificate against P_{i^*}; (2) is true.

Proving invariant $\mathcal{I}$. To prove invariant $\mathcal{I}$ we make the following arguments:

1. We first argue that the honest parties retrieve solvable puzzles $\overrightarrow{\mathsf{puz}} = \mathsf{puz}_1, \ldots, \mathsf{puz}_{j^*}, \ldots \mathsf{puz}_n$. Indeed, if this was not the case the honest parties can produce a certificate $\mathsf{cert} = (1, \mathsf{aux})$ with witch they can accuse P_{i^*} because they receive a signatures of $\overrightarrow{\mathsf{puz}}$ from P_{i^*}. Otherwise P_{i^*} aborts before signing the puzzles. In this case, the abort probability is independent from the probability of cheating: it follows from the UC-security of Π_{check}, the hiding of commitment scheme, the security of encryptions and the security of time lock puzzles that P_{i^*} does not know which execution will be checked.
2. If the encryptions sent by P_{i^*} correspond to malformed messages, then since the encryption scheme enjoys correctness and the honest parties obtains signature of P_{i^*} on her ciphertexts, the honest parties can produce a certificate

$\mathsf{cert}(3, \mathsf{aux})$ for accusing P_{i^*}. Note that if P_{i^*} aborts before signing the ciphertexts the probability of abort is independent from the probability of cheating for the same reasons explained above.

3. If the decommitment of C^{share} sent by P_{i^*} are not valid then, since the commitment scheme enjoys correctness and the honest parties obtains signature on C^{share}, the honest parties can produce a certificate $\mathsf{cert}(2, \mathsf{aux})$ for accusing P_{i^*}. Note that if P_{i^*} aborts before signing the ciphertexts the probability of abort is independent from the probability of cheating for the same reasons explained above.
4. From observations 1, 2 and 3 follow that if the protocol did not terminate then P_{i^*} encrypted valid messages, and valid decommitments of C^{share} that are retrievable from solvable puzzle but it can still be the case that decommitments of C^{rnd} and the partial certificates are not retrievable from $\overrightarrow{\mathsf{puz}}$. We now ague that if the protocols continue until step 14 then decommitments of C^{rnd} and the partial certificates are retrievable from $\overrightarrow{\mathsf{puz}}$. We make the following arguments:
   (a) If P_{i^*} commits to invalid shares in C^{share} the honest parties abort. C^{share} are computed before it is revealed which executions are been checked. It follows from the UC-security of Π_{check}, the hiding of C^{share} and the security of the puzzle system that the probability of abort is independent from the probability of cheating.
   (b) From the UC-security of Π_{check} we can claim that, unless of negligible probability, the only way that P_{i^*} can cheat in this point is using malformed inputs to Π_{check}. If this is case by the definition of the function computed by Π_{check} (i.e., authenticate secret sharing) and the binding of C^{share} we are ensured that the honest parties abort. Roughly speaking this cheating strategy translates to a P_{i^*} that commits to invalid shares in C^{share}, so we can follow the same analysis of point 4a.

□

k	Protocol	Preprocessing			Online	Total	
		# OTs	R-OT	S-OT		R-OT	S-OT
2	Passive [8]	8.00	128.80	0.90	256.21	385.01	257.11
	Passive (ours)	6.01	96.81	0.68	256.21	353.01	256.88
	Covert (non-id)	12.00	193.61	1.35	256.21	449.82	257.56
	Active	54.00	576.30	9.05	256.21	832.51	265.26
4	Passive [8]	8.00	128.80	0.90	256.21	385.01	257.11
	Passive (ours)	6.01	96.81	0.68	256.21	353.01	256.88
	Covert (non-id)	24.00	387.22	2.70	256.21	643.43	258.91
	Active	54.00	576.30	9.05	256.21	832.51	265.26
6	Passive [8]	8.00	128.80	0.90	256.21	385.01	257.11
	Passive (ours)	6.01	96.81	0.68	256.21	353.01	256.88
	Covert (non-id)	36.00	580.83	4.06	256.21	837.04	260.26
	Active	54.00	576.30	9.05	256.21	832.51	265.26
8	Passive [8]	8.00	128.80	0.90	256.21	385.01	257.11
	Passive (ours)	6.01	96.81	0.68	256.21	353.01	256.88
	Covert (non-id)	48.00	774.44	5.41	256.21	1030.65	261.61
	Active	54.00	576.30	9.05	256.21	832.51	265.26
10	Passive [8]	8.00	128.80	0.90	256.21	385.01	257.11
	Passive (ours)	6.01	96.81	0.68	256.21	353.01	256.88
	Covert (non-id)	60.00	968.06	6.76	256.21	1224.26	262.96
	Active	54.00	576.30	9.05	256.21	832.51	265.26
12	Passive [8]	8.00	128.80	0.90	256.21	385.01	257.11
	Passive (ours)	6.01	96.81	0.68	256.21	353.01	256.88
	Covert (non-id)	72.00	1161.67	8.11	256.21	1417.87	264.32
	Active	54.00	576.30	9.05	256.21	832.51	265.26

Table 2. Bandwidth costs (in MB) and OT costs (millions of OTs) for passive, covert and actively secure protocols without identifiable abort, in a Boolean circuit with 100 thousand ANDs, with different values of the replication factor k