Reverse Firewalls for Actively Secure MPCs

Suvradip Chakraborty[⋆], Stefan Dziembowski[⋆⋆], and Jesper Buus Nielsen[⋆⋆⋆]

Abstract. Reverse firewalls were introduced at Eurocrypt 2015 by Miro-nov and Stephens-Davidowitz, as a method for protecting cryptographic protocols against attacks on the devices of the honest parties. In a nutshell: a reverse firewall is placed outside of a device and its goal is to "sanitize" the messages sent by it, in such a way that a malicious device cannot leak its secrets to the outside world. It is typically assumed that the cryptographic devices are attacked in a "functionality-preserving way" (i.e. informally speaking, the functionality of the protocol remains unchanged under this attacks). In their paper, Mironov and Stephens-Davidowitz construct a protocol for passively-secure two-party computations with firewalls, leaving extension of this result to stronger models as an open question.

In this paper, we address this problem by constructing a protocol for secure computation with firewalls that has two main advantages over the original protocol from Eurocrypt 2015. Firstly, it is a *multi*party computation protocol (i.e. it works for an arbitrary number n of the parties, and not just for 2). Secondly, it is secure in much stronger corruption settings, namely in the *actively corruption model.* More precisely: we consider an adversary that can fully corrupt up to $n-1$ parties, while the remaining parties are corrupt in a functionality-preserving way.
Our core techniques are: malleable commitments and malleable non-interactive zero-knowledge, which in particular allow us to create a novel protocol for multiparty augmented coin-tossing into the well with reverse firewalls (that is based on a protocol of Lindell from Crypto 2001).

1 Introduction

The traditional approach to cryptography is to design schemes in a black-box way, i.e, under the assumption that the devices that execute cryptographic algorithms are fully trusted. Abstract, "black-box" cryptography is currently well-understood, and there exist several algorithms that implement basic cryptographic tasks in a way that is secure against a large class of attacks (under very plausible assumptions). Therefore, one can say that cryptographic algorithms, if implemented correctly, are the most secure part of digital systems.

Unfortunately, once we get closer to the real "physical world" the situation becomes much less satisfactory. This is because several real-life attacks on cryptographic devices are based on attacking the *implementation*, not the abstract mathematical algorithm. In particular, the adversary can sometimes *tamper* with the device and change the way in which it behaves (e.g. by installing so-called "Trojan horses" on it). What can be viewed as the extreme case of the tampering attacks are scenarios in which the device is produced by an adversarial manufacturer, who maliciously modifies its design.

[⋆] Institute of Science and Technology Austria. suvradip.chakraborty@ist.ac.at
[⋆⋆] University of Warsaw. s.dziembowski@crypto.edu.pl
[⋆⋆⋆] Aarhus University. jbn@cs.au.dk

Such attacks are quite realistic, since, for the economical reasons, private companies and government agencies are often forced to use hardware that they did not produce themselves. Another source of such attacks are the insiders that originate from within a given company or organization. Last but not least, some attacks of this type can originate from the governments. The revelations of Edward Snowden disclosed a massive scale of the US government cyberattacks directed against the individuals (both within the US and abroad). It is generally believed that many other governments take similar actions, one recent example being the "Chinese hack chip" attack (revealed in October 2018) that reached almost 30 U.S. companies, including Amazon and Apple.

Countermeasures. Starting from late 1990s there has been a significant effort in the cryptographic community to address this kind of "implementation attacks", by extending the black-box model to cover also them (see, e.g., [24, 22]). More recently, Mironov and Stephens-Davidowitz [25] put forward another method that they called *reverse firewalls*. On a high level (for a formal definition see Sec. 3), this technique addresses the problem of information leakage from cryptographic implementations that are malicious, either because they were produced by an adversarial manufacturer, or because they are were maliciously modified at a later stage. More concretely, reverse firewalls are used to protect against attacks in which a malicious implementation leaks some of its secrets via so-called "subliminal channels" [27], i.e, by encoding this secrets in innocently-looking protocol messages. In a nutshell, a reverse firewall is an external device that is put between a party P and the external world in order to "sanitize" the messages that are sent and received by P. A reverse firewall is *not* a trusted third party, and, in particular, it cannot be used to keep P's secrets and to perform operations "in P's name". Reverse firewalls come in different variants. The most popular one, that we also consider in this paper, requires that the reverse firewalls provide protection only against the aforementioned "informational leakage" attack (and not against attacks that may influence the output of the computation). In particular, in this model, we are *not* concerned with the *correctness* of the computation. More formally, we assume that all the adversarial tampering cannot change the functionality of the entire protocol. This type of attacks are called "*functionality maintaining*" corruptions [25]. The authors of [25] provide a construction of a two-party passively secure computation protocol with a reverse firewall, leaving the generalization of this construction to stronger security notions as an open problem. Reverse firewalls has been recently used in a very practical context by Dauterman et al. in [11] in a design of a *True2F* system that is based on a firewalled key generation and ECDSA signature generation. One of the potential applications of this system are the cryptocurrency wallets.

Our contribution. We address the open problem of [25] by providing a construction of reverse firewalls for secure computation in a much stronger security model, and in a more general setting. More concretely, we show a solution to the problem by constructing *multi*party computation protocols with *active* security. Recall that in the active security settings the corrupt parties can misbehave in an arbitrary way, i.e., the adversary takes a full control over them, and, besides learning their inputs, can instruct them to take any actions of his choice. It is well known [18, 17] that such protocols can be constructed even if a majority of parties is corrupt (assuming that no *fairness* is guaranteed, i.e., the adversary can prevent the honest parties from learning their outputs, after she learns the outputs of corrupt parties). In this work, we show an MPC protocol (based on [18, 17]), together with a reverse firewall for it, that provides security in a very strong sense: it can tolerate up to $n - 1$ "standard" (active) corruptions (where n is the number of parties) plus a corruption of the remaining parties, as long as it is "functionality maintaining" and these parties are protected by their corresponding reverse firewalls. The core technique that we use in this construction is a novel protocol for multiparty augmented parallel coin-tossing into the well with reverse firewalls (our starting point for this construction is a protocol of Lindell [23]).

Our result shows the general feasibility of MPCs with reverse firewalls. While we do not focus on concrete applications, we believe that our approach can lead to some practical concrete constructions, especially in the light of [11] (see above). For example, to further increase the security of hardware wallets (e.g. in critical applications such as cryptocurrency exchanges), one could develop reverse firewalls for threshold ECDSA (see, e.g., [16, 13]). Our results show that this is in principle possible, but further work to bring these ideas to practice is needed.

Other related work. After the publication of [25] there has been some follow-up work on the reverse firewalls. In particular [12] constructed a firewalled protocol for CCA-secure message transmission, and [9] provide protocols for oblivious signature-based envelopes with firewalls, and oblivious transfer (this is done using a new technique called "malleable smooth projective hash function" that they develop in this paper). In [3] Ateniese et al. use reverse firewalls to construct signature schemes secure against arbitrary tampering. Reverse firewalls are also related to several earlier topics in cryptography such as the algorithm-substitution attacks, subliminal channels and divertible protocols, combiners, kleptography, collusion-free protocols and mediated collusion-free protocols and more. Due to space constraints, we refer the reader to Sec. 1.1 of [25] for an overview of these topics and their relation to reverse firewalls.

1.1 Overview of our construction

On a high level, our construction can be viewed as "adding reverse firewalls to the MPC protocol of [18, 17]". In particular, we follow the protocol structure presented in Sec. 3.3.3 of [17], i.e.: the parties generate random strings to which they are committed (this is called "augmented coin-tossing"), they commit to their inputs (the "input commitment protocol"), and finally they perform the "authenticated computation" in which they do computations on these values, simultaneously proving (in zero knowledge) that the computation is done correctly (in our construction we use a non-interactive version of zero-knowledge protocols, NIZKs, [6]). The main things that need to be addressed in adding reverse firewalls to this protocol is to construct protocols for commitment schemes and NIZKs with firewalls (since the correctness of every step of the computation is proven in zero knowledge, we do not need to construct separate firewalls for the computations itself). Essentially, these firewalls are constructed by "re-randomizing" the messages that are sent by the parties. More precisely: for messages that come from commitments, we exploit the standard homomorphic properties of such schemes, and for NIZKs we use the "controlled-malleable NIZK proof systems" of [8][1]. On a high level, the firewalls can re-randomize a protocol transcript exploiting homomorphic properties of the commitment scheme, and controlled malleability property of the NIZK proofs (where the controlled malleability is "tied" to the appropriate mauling of the commitments). One of the key ingredients of our construction is a firewalled scheme for augmented coin tossing. This is built by combining the firewalled protocols for commitment and NIZKs with the coin-tossing protocol of Lindell [23].

Reverse firewalls for multi-party (augmented) coin-tossing. Let us explain the design principle of our reverse firewall for the multi-party augmented coin-tossing protocol in more details. The starting point of our protocol is the 2-party augmented parallel coin-tossing of Lindell [23]. The protocol of [23] uses a "commit-and-prove" technique, where one party (often called the initiating party) commits to a random bit-string and proves in zero-knowledge about the consistency of the committed value. The other party also sends a random bit-string to this party. The final string is the exclusive OR of both these strings and the initiating party commits to this final string. The protocol ends by outputting a random bit-string (which the initiating party gets), and the commitment value to the final bit-string (which the other party receives). First, we extend this protocol to the multi-party setting, and then design a reverse firewall for this protocol. We assume that the honest parties are corrupted in a functionality-maintaining way. Note that, in the traditional model of corruption the adversary completely controls the party and may also cause the party to deviate arbitrarily from the protocol. In contrast, functionality-maintaining corruptions also allows the adversary to completely control the party and also cause the party to deviate from the protocol specification as long as it *does not* violate or break the functionality (i.e, correctness) of the underlying protocol. The first observation is that the corrupted parties may not necessarily commit to a random bit-string. Even if it does so, the commitment may also leak information about the committed value (say the randomness used to commit may leak additional information about the bit-string). Secondly, the bit-strings sent by the other parties to the initiating party may also act as a subliminal channel to leak secret information.

The main idea behind our firewall design is that it should somehow be possible to maul the commitment in such a way that the committed element is random (even if the initial bit-string is not chosen randomly) and the commitment is itself re-randomizable (so that the commitment appears to be "fresh"). For this, we assume the commitment scheme to be additively *homomorphic* (with respect to an appropriate relation), which suffices for our purpose. At this point, the original zero-knowledge proof (that conforms to the initial commitment) is no longer *valid* with respect to the mauled commitment. Hence, the firewall needs a way to appropriately maul the proof (so that the mauled proof

[1] Since we use a NIZK proof system, we need to assume a trusted setup algorithm which generates a common reference string (CRS) to be used by all the parties. We assume that the CRS is hardwired inside the code of each party.

is consistent with the mauled commitment), and also to re-randomize the proof (so that the randomness used to proof does not leak any information on the witness, which is the committed string). To this end, we use the controlled-malleable NIZK proof systems (cm-NIZK) introduced by Chase et al. [8]. We replace the (interactive) zero-knowledge proofs used in the protocol of [23] with cm-NIZK proofs (with a trusted setup procedure). The firewall then re-randomizes the shares (bit-strings) of the other parties in such a way that is consistent with the initial mauling of the commitment and the proof.

However, at this point another technical difficulty arises: the views of all the parties are not identical– in particular, the view of the initiating party and the other parties are not same, due to the above mauling by the firewall. While this appears to be problematic as far as the functionality of the protocols is concerned, we show that the firewall can again re-maul the transcript in such a way that the views of all the parties become consistent, without compromising on the security of the protocol. Here by "consistent" we mean that the initiating party (of the coin-tossing protocol) receives a random bit string, and the rest of the parties receive the commitment to *the same* bit-string.

Indeed, we show that at the end the initiating party ends up with a random bit-string (as required by the functionality), even if it is corrupted (in a functionality-maintaining way) and the other parties obtains a secure commitment to this bit-string. We show that the above firewall maintains functionality, preserves security for the honest parties, and also provide weak exfiltration-resistant[2] against other parties. Finally, we stress that the above mauling operations, specially the mauling of the NIZK proofs, does not require the firewall to know the original witness (chosen by the initiating party), which makes it interesting and doable from the firewall perspective (since it shares no secret with any of the parties). We refer the reader to Sec. 6.1 for the details.

Reverse firewalls for other protocols. We also design reverse firewalls for the multi-party input commitment protocol and the multi-party authenticated computation protocol, which are also used as key ingredients for our final actively-secure MPC protocol. The reverse firewalls for these protocols are relatively much simpler and involve only re-randomizing the commitment and the NIZK proof (in case of the input commitment protocol) and re-randomizing the proof (for the authenticated computation protocol). We show that both the firewalls corresponding to these two protocols preserve security and is exfiltration-resistant against other parties.

The final compiler. Finally, we show the construction of our actively-secure MPC protocols in the presence of reverse firewalls. Our final compiler is similar to the compiler presented in [17], however, adapted to the setting of reverse firewalls. The compiler takes as input any semi-honest MPC protocol (without reverse firewalls) and runs the multi-party input commitment protocol, the multi-party (augmented) coin-tossing protocol and the multi-party authenticated computation protocol in the reverse firewall setting (in sequential order) to obtain the final actively-secure MPC protocol. On a high level,

[2] Informally, the exfiltration-resistant property stipulates that the corrupt implementation of party does not leak any information through the firewall. Weak exfiltration-resistance guarantees the same property when the party is corrupted in a functionality-maintaining way (and not arbitrarily).

after the input commitment and the coin-tossing protocol (in the presence of reverse firewalls) the inputs and the random pads of all the (honest) parties are fixed. Now, since the honest parties are corrupted in a functionality-maintaining way, the computation performed by the party in the authenticated computation protocol is determined, and the final zero-knowledge proofs conform to these computations. Hence, at this point, the security of the underlying semi-honest MPC protocol (without using reverse firewalls) can be invoked to argue security of our final actively-secure MPC protocol (in the presence of reverse firewalls).

Compiler for reverse firewalls for broadcast model. As a contribution of independent interest, we also present a compiler for reverse firewalls (RF) in the broadcast model. In particular, existence of a broadcast channels in the RF setting is a stronger assumption than the existence of a broadcast channel in the classical setting. To this end, we present a version of the Dolev Strong protocol [14] secure in the RF setting. The key idea is to transform the original Dolev Strong protocol to be a "unique message protocol", so that, at any given point there is only one possible message that a party can send. We implement this by replacing the signatures in the Dolev-Strong protocol with *unique signatures*. Intuitively this works because: on any input in the Dolev-Strong protocol, the only allowed message consists of adding a signature on a well-defined message. The signature is either sent or added to a valid set. Since the signatures are unique and the parties are corrupted in a functionality-maintaining way, it is forced to send the unique message at that particular round. In general, the above idea also works if we replace the signatures in the Dolev-Strong protocol with *re-randomizable* signatures [21, 28]. Note that unique signatures are efficiently re-randomizable. We note that, our result also nicely complements the result of Ateniese et al. [3], who gave a negative result for the construction of RF for arbitrary signature schemes. On the positive side, they show constructions of RF for the class of re-randomizable signature schemes (which includes unique signatures as well).

Constructing actively secure MPC from semi-malicious MPC in the RF setting. A recent line of work [26, 5, 15] constructs 2-round MPC protocols achieving *semi-malicious* security, which means that the protocol is secure for all (possibly adversarial) choices of the random coins of the parties. Furthermore, following the compilation paradigm of [2, 1], one can immediately obtain maliciously secure Universal Composable (UC) MPC protocols in the CRS model, using NIZK proofs. At first thought, it seems that if we start with any of these 2-round semi-malicious MPC protocols and use a controlled-malleable NIZK proof on top (instead of just NIZK) we can hope to get a 2-round actively secure MPC protocol in the RF setting. However, this approach does not work: semi-malicious security protects the other parties against a semi-malicious corrupted party, but does not protect the corrupted party itself. In fact, a maliciously chosen random tape might be used to leak information covertly, so semi-malicious security does not provide *exfiltration resistance.*

On the Trusted Setup assumption. Our construction of the actively secure MPC protocol uses a controlled-malleable NIZK (cm-NIZK) proof, and hence is in the CRS model. This is in contrast to the original GMW protocol [18] which does not require any trusted setup assumption, since it uses interactive zero-knowledge proofs. A natural idea is whether it is possible to replace the cm-NIZK proofs with *controlled-malleable interactive* ZK (cm-IZK) proofs. Indeed, while it is not hard to see that one can construct cm-IZK proofs from one-way functions[3], it seems that the techniques of our paper are unlikely to extend to work with cm-IZK proofs. The main challenge is in making the views of the parties consistent in the final MPC protocol. We consider this as an interesting open problem to remove the trusted setup assumption.

Organization of the paper. The basic definitions and notation are provided in Sect. 2 (Sect. 3 contains the definitions related to the reverse firewalls). Our main technical contribution is presented in Sect. 4, with Sections 5 and 5.1 describing the construction of broadcast channels in the reverse firewall setting, Sections 6—6.4 describing the ingredients of our construction, and Sect. 6.5 putting them together into a single "protocol compiler" algorithm. The security of our construction in stated and proven in Thm. 7.

2 Preliminaries

In this section we introduce some standard notation and terminology that will be used throughout the paper. For an integer $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, 2, \cdots, n\}$ and for any pair of integers $1 < i < j \leq n$, we denote by $[i, j]$ the set $\{i, i+1, \cdots, j\}$. For a distribution or random variable X, we denote $x \leftarrow X$ the action of sampling an element x according to X. For any integer $m \in \mathbb{N}$, we write U_m to denote the uniform distribution over all m-bit strings. A decision problem related to a language $L \subseteq \{0,1\}^*$ requires to determine if a given string x is in L or not. In this paper, we consider $\mathcal{NP}$ language. Corresponding to each $\mathcal{NP}$ language L, we can associate a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ defining L such that: $L = \{x : \exists \omega \text{ s.t. } (x, \omega) \in R\}$ and $|\omega| \leq poly(|x|)$. We call x the *statement/theorem*, and ω the *witness* testifying the membership of x in the language L, i.e., $x \in L$. Let $T = (T_x, T_\omega)$ be a pair of efficiently computable n-ary functions, $T_x : \{\{0,1\}^*\}^n \rightarrow \{0,1\}^*$. We call such a tuple T as an n-ary transformation. Following [8], we define what it means for a transformation $T = (T_x, T_w)$ to be *admissible* with respect to a NP relation R.

Definition 1. (Admissible transformations [8]). *An n-ary transformation $T = (T_x, T_w)$ is said to* admissible *for an efficient relation R, if R is* closed *under T, i.e, for any n-tuple $\{(x_1, \omega_1), \cdots, (x_n, \omega_n)\} \in R^n$, it holds that the pair $\big(T_x(x_1, \cdots x_n), T_\omega(\omega_1, \cdots, \omega_n)\big) \in R$. We say that a class or set of transformations $\mathcal{T}$ is an* allowable *set of transformation if every transformation $T \in \mathcal{T}$ is admissible for R.*

[3] One can modify the zero-knowledge proofs for the Graph Hamiltonicity problem or the 3-Coloring problem to obtain cm-IZK proofs by replacing the commitments with homomorphic commitments, similar to our coin-tossing protocol

2.1 Homomorphic Commitments

A (non-interactive) commitment scheme consists of three polynomial time algorithms $(\mathcal{G}, K, \mathsf{com})$. The probabilistic setup algorithm $\mathcal{G}$ takes as input the security parameter λ and outputs the setup parameters par. The key generation algorithm K is a probabilistic algorithm that takes as input par and generates a commitment key ck. We assume that the commitment key ck includes the description of the message space $\mathcal{M}$, the randomness space $\mathcal{R}$ and the commitment space $\mathcal{C}$ to be used in the scheme. We also assume it is possible to efficiently sample elements from $\mathcal{R}$. The algorithm com takes as input the commitment key ck, a message m from the message space $\mathcal{M}$ and "encodes" m to produce a commitment string c in the commitment space $\mathcal{C}$. Additionally, we also require the commitment scheme to be *homomorphic* [19, 20], i.e, we assume that $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{C}$ are groups with the *homomorphic* property, and if we add any two commitments, the resulting commitment will encode the sum of the underlying messages. The formal definition follows:

Definition 2 (Homomorphic commitments [19]). *A homomorphic trapdoor commitment scheme consists of the tuple of algorithms* $(\mathcal{G}, K, \mathsf{com})$ *as described above, with the security properties as stated below:*

- **(Perfect hiding).** *The triple* $(\mathcal{G}, K, \mathsf{com})$ *is* perfectly hiding *if for all stateful adversaries* $\mathcal{A}$*, we have:*

$$\Pr\left[b' = b \,\middle|\, \begin{array}{c} \mathsf{par} \leftarrow \mathcal{G}(1^\lambda); \mathsf{ck} \leftarrow K(\mathsf{par}); (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{par}, \mathsf{ck}); \\ b \xleftarrow{\$} \{0,1\}; c \leftarrow \mathsf{com}_{\mathsf{ck}}(m_b); b' \leftarrow \mathcal{A}(c, \mathsf{st}) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

- **(Computationally binding).** *The triple* $(\mathcal{G}, K, \mathsf{com})$ *is* computationally binding *if for all non-uniform polynomial time stateful adversaries* $\mathcal{A}$*, we have:*

$$\Pr\left[\begin{array}{c} m \neq m' \wedge \\ m, m' \neq \perp \end{array} \,\middle|\, \begin{array}{c} \mathsf{par} \leftarrow \mathcal{G}(1^\lambda); \mathsf{ck} \leftarrow K(\mathsf{par}); (m, m', r, r') \leftarrow \mathcal{A}(\mathsf{par}, \mathsf{ck}) : \\ \mathsf{com}_{\mathsf{ck}}(m; r) = \mathsf{com}_{\mathsf{ck}}(m'; r') \end{array} \right] \leq \mathsf{negl}(\lambda)$$

- **(Homomorphic).** *The commitment scheme* $(\mathcal{G}, K, \mathsf{com})$ *is* homomorphic *if* K *always outputs* ck *describing groups* $\mathcal{M}$*,* $\mathcal{R}$*,* $\mathcal{C}$*, which are written additively, such that for all* $m, m' \in \mathcal{M}$*,* $r, r' \in \mathcal{R}$ *we have:* $\mathsf{com}_{\mathsf{ck}}(m; r) + \mathsf{com}_{\mathsf{ck}}(m'; r') = \mathsf{com}_{\mathsf{ck}}(m + m'; r + r')$*.*

2.2 Controlled Malleable Non-Interactive Zero-Knowledge Proofs.

We recall the definitions of controlled-malleable non-interactive proof systems from [8]. A non-interactive proof system for a $\mathcal{NP}$ language L associated with relation R consists of three (probabilistic) polynomial-time algorithm $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V})$. The Common Reference String (CRS) generation algorithm CRSGen takes as input the security parameter 1^λ, and outputs CRS σ_{crs}. The prover algorithm P takes as input σ_{crs}, and a pair $(x, \omega) \in R$, and outputs a proof π. The verifier algorithm V takes as input σ_{crs}, a statement x and a purported proof π, and outputs a decision bit $b \in \{0, 1\}$, indicating whether the proof π with respect to statement x is accepted or not (with 0 indicating reject, else accept). The two most basic requirements from such a proof system are *perfect completeness* and *adaptive soundness* with respect to (possibly unbounded) cheating provers. Besides, we also want the NIZK proof systems for efficient relations R that are (1) *malleable* with respect to an allowable set of transformations $\mathcal{T}$, i.e., for any $T \in \mathcal{T}$, given proofs $\pi_1, \cdots, \pi_n$ for statements $x_1, \cdots, x_n \in L$ they can be transformed into a proof π for the statement $T_x(x_1, \cdots, x_n)$, and (2) *derivation private*, i.e. the resultant proof π cannot be distinguished from a fresh proof computed by the prover on input $\big(T_x(x_1, \cdots, x_n), T_\omega(\omega_1, \cdots, \omega_n)\big)$. We also want zero-knowledge property and simulation-sound extractability property to hold for the NIZK proof system under controlled malleability, as defined below.

Definition 3. (Controlled-malleable NIZK proof system [8]). A *controlled malleable non-interactive* (cm-NIZK) *proof system* for a language L associated with a $\mathcal{NP}$ relation R consists of four (probabilistic) polynomial-time algorithms $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V}, \mathsf{ZKEval})$ such that the following conditions hold:

- (*Completeness*). For all $\sigma_{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$, and $(x, \omega) \in R$, it holds that $\mathsf{V}(\sigma_{crs}, x, \pi) = 1$ for all proofs $\pi \leftarrow \mathsf{P}(\sigma_{crs}, x, \omega)$.
- (*Soundness*). We say that $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V})$ satisfies *adaptive soundness* if for all PPT (malicious) provers P^* we have:

$$\mathsf{Pr}\Big[\sigma_{crs} \leftarrow \mathsf{CRSGen}(1^\lambda); (x, \pi) \leftarrow \mathsf{P}^*(\sigma_{crs}) : \mathsf{V}(\sigma_{crs}, x, \pi) = 0 \text{ if } x \notin L\Big] > 1 - \mathsf{negl}(\kappa).$$

  for some negligible function $\mathsf{negl}(\kappa)$. Perfect soundness is achieved when this probability is always 1.
- (*Malleability*). Let $\mathcal{T}$ be a set of allowable transformation for an efficient relation R. Then the proof system $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V})$ is said to be *malleable* with respect to $\mathcal{T}$, if there exists an efficient algorithm ZKEval that does the following: ZKEval takes as input σ_{crs}, the description of a n-ary admissible transformation $T \in \mathcal{T}$, statement-proof pairs (x_i, π_i), where $1 \leq i \leq n$, such that $\mathsf{V}(\sigma_{crs}, x_i, \pi_i) = 1$ for all i, and outputs a proof π for the statement $x = T(\{x_i\})$ such that $\mathsf{V}(\sigma_{crs}, x, \pi) = 1$.
- (*Rerandomizability*). We say that the NIZK proof system $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V})$ for relation R is re-randomizable if there exists an additional algorithm $\mathsf{RandProof}$, such that the probability of the event that $b' = b$ (where $b \overset{\$}{\leftarrow} \{0, 1\}$ is sampled uniformly at random) in the following game is negligible:
  - $\sigma_{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$.
  - $\big(\mathsf{state}, x, w, \pi\big) \overset{\$}{\leftarrow} \mathcal{A}(\sigma_{crs})$.
  - If $\mathsf{V}(\sigma_{crs}, x, \pi) = 0$, or $(x, w) \notin R$, output $\perp$. Otherwise form

$$\pi' \leftarrow \begin{cases} \mathsf{P}\big(\sigma_{crs}, x, w\big) & \text{if } b = 0 \\ \mathsf{RandProof}(\sigma_{crs}, x, \pi) & \text{if } b = 1. \end{cases}$$

  - $b' \leftarrow \mathcal{A}(\sigma_{crs}, \pi')$

- (*Derivation privacy*). We say that the NIZK proof system $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V}, \mathsf{ZKEval})$ for relation R with respect to $\mathcal{T}$ is *derivation-private*, if for all adversaries $\mathcal{A}$ and bit b, the probability $p_b^{\mathcal{A}}(\lambda)$ that the event $b' = b$ (where $b \xleftarrow{\$} \{0,1\}$ is sampled uniformly at random) in the following game is negligible:
  - $\sigma_{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$.
  - $\big(\mathsf{state}, (x_1, \omega_1, \pi_1), \cdots, (x_q, \omega_q, \pi_q), T\big) \leftarrow \mathcal{A}(\sigma_{crs})$.
  - If $\mathsf{V}(\sigma_{crs}, x_i, \pi_i) = 0$ for some i, $(x_i, \omega_i) \notin R$ for some i, or $T \notin \mathcal{T}$, abort and output $\perp$. Otherwise compute,

$$\pi \leftarrow \begin{cases} \mathsf{P}\big(\sigma_{crs}, T_x(x_1, \cdots, x_q), T_\omega(\omega_1, \cdots, \omega_q)\big) & \text{if } b = 0 \\ \mathsf{ZKEval}(\sigma_{crs}, T, \{(x_i, \pi_i)\}_{i \in [q]}) & \text{if } b = 1. \end{cases}$$

  - $b' \leftarrow \mathcal{A}(\mathsf{state}, \pi)$.

- (*Controlled-malleable simulation-sound extractability*). Let $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V})$ be a NIZK proof of knowledge (NIZKPoK) system for the relation R, with a simulator $(\mathcal{S}_1, \mathcal{S}_2)$ and an extractor $(\mathcal{E}_1, \mathcal{E}_2)$. Let $\mathcal{T}$ be an allowable set of unary transformation for the relation R such that membership in $\mathcal{T}$ is efficiently testable. Let $\mathcal{SE}_1$ be an algorithm, that on input 1^λ outputs $(\sigma_{crs}, \tau_s, \tau_e)$ such that (σ_{crs}, τ_s) is distributed identically to the output of $\mathcal{S}_1$. Consider the following game with the adversary $\mathcal{A}$:
  - $(\sigma_{crs}, \tau_s, \tau_e) \leftarrow \mathcal{SE}_1(1^\lambda)$.
  - $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_2(\sigma_{crs}, \tau_s, \cdot)}(\sigma_{crs}, \tau_e)$.
  - $(\omega, x', T) \leftarrow \mathcal{E}_2(\sigma_{crs}, \tau_e, x, \pi)$.

  We say that the NIZKPoK satisfies *controlled-malleable simulation-sound extractability* (CMSSE) if for all *PPT* algorithms $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that the probability that $\mathsf{V}(\sigma_{crs}, x, \pi) = 1$ and $(x, \pi) \notin \mathcal{Q}$ (where $\mathcal{Q}$ is the set of queried statements and their responses) but either (1) $\omega \neq \perp$ and $(x, \omega) \notin R$; (2) $(x', T) \neq (\perp, \perp)$ and either $x' \notin \mathcal{Q}_x$ (the set of queried instances), $x \neq T_x(x')$, or $T \notin \mathcal{T}$; (3) $(\omega, x', T) = (\perp, \perp, \perp)$ is at most $\nu(\lambda)$.

Theorem 1. [8] *If a proof system is both malleable and randomizable and uses* $\mathsf{ZKEval}' = \mathsf{RandProof} \circ \mathsf{ZKEval}$*, then it is also derivation private.*

Remark 1. The definition of CM-SSE is a weakening of the definition of (standard) simulation-sound extractability (SSE). The notion of CM-SEE intuitively says that the extractor will either extract a valid witness ω corresponding to the new statement x (as in SSE), or a previously proved statement x' and a transformation T in the allowable set $\mathcal{T}$ that could be used to transform x' into the new statement x. Note that, when $\mathcal{T} = \emptyset$, we obtain the standard notion of SSE-NIZK as defined by Groth [20]. However, as shown in [8], this definitional relaxation is necessary, since the standard notion of SSE is impossible to achieve for malleable proof systems.

2.3 Secure computation

We present the definition of general multi-party computation protocols (for an introduction to this topic see, e.g., [10]). We follow the definitions as presented in [17, 23], which in turn follows the definitions of [7, 4].

Multi-party protocols. Let n denote the number of parties involved in the protocol. We assume that n is fixed. A multi-party protocol problem is casted by specifying a random process which maps sequences of inputs (one input per each of the n parties) to sequences of outputs (one for each of the n parties). We refer to such a process as a n-ary *functionality*, denoted by $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$, where $f = (f_1, \cdots, f_n)$. For a input vector $\boldsymbol{x} = \{x_1, \cdots, x_n\}$ the output is a tuple of random variables denoted by $(f_1(\boldsymbol{x}), \cdots, f_n(\boldsymbol{x}))$. The i^{th} party P_i initially holds the input x_i and obtains the i^{th} element in $f(x_1, \cdots, x_n)$, i.e. $f_i(x_1, \cdots, x_n)$. We also assume that all the parties hold input of equal length, i.e., $|x_i| = |x_j|$ for all $i, j \in [n]$. We will denote such a functionality as $(x_1, \cdots, x_n) \mapsto (f_1(x_1, \cdots x_n), \cdots, f_n(x_1, \cdots x_n))$.

Adversarial behavior. For the analysis of our protocols we consider the *malicious* adversarial model. A malicious adversary may corrupt a subset of parties and can completely control these parties and deviate arbitrarily from the specified protocol. We assume a *static* corruption model, where the set of corrupted or dishonest parties are already specified before the execution of the protocol. A weaker model of security is the *semi-honest* model, where the adversary has to follow the protocol as per its specification, but it may record the entire transcript of the protocol to infer something beyond the output of the protocol. We consider the definition of security in terms of a real-world and ideal-world simulation paradigm, as in [17]. In the ideal model, we assume the existence of an in-corruptible trusted third party (TTP). In the semi-honest model, all the parties send their local inputs to the TTP, who computes the desired functionality and send back the prescribed outputs to them. The honest parties then output their respective outputs, while the semi-honest parties output an arbitrary probabilistic polynomial-time function of their respective inputs and the outputs obtained from the TTP. In contrast, in the malicious model the malicious parties may substitute their local input and send it to the TTP in the first place. We assume that the TTP always answers the malicious parties first. The malicious parties may also abort the execution of the protocol by refraining from sending their own messages. Finally, as in the semi-honest model, each honest party outputs its output as received from the TTP, while the malicious parties may output an arbitrary probabilistic polynomial-time function of their initial inputs and the outputs obtained from the TTP.

Definition 4 (Malicious adversaries–the ideal model). *Let $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$ be a n-ary functionality as defined above. Let $\mathcal{I} = \{i_1, \cdots, i_q\} \subset [n]$, and $(x_1, \cdots, x_n)_I = (x_{i_1}, \cdots, x_{i_q})$. A pair $(\mathcal{I}, \mathcal{C})$ where $\mathcal{I} \subset [n]$ and $\mathcal{C}$ is a polynomial-size circuit family represents an adversary in the ideal model. The joint execution under $(\mathcal{I}, \mathcal{C})$ in the ideal model (on input sequence $\boldsymbol{x} = (x_1, \cdots, x_n)$), denoted by $\mathsf{IDEAL}_{f,(\mathcal{I},\mathcal{C})}(\boldsymbol{x})$ is defined as follows:*

$$\begin{array}{ll} (\mathcal{C}(\boldsymbol{x}_{\mathcal{I}}, \bot), \bot, \cdots, \bot) & \textit{if } \mathcal{C}(\boldsymbol{x}_{\mathcal{I}}) = \bot\ . \\ (\mathcal{C}(\boldsymbol{x}_{\mathcal{I}}, f_I(\mathcal{C}(\boldsymbol{x}_{\mathcal{I}}), \boldsymbol{x}_{\bar{\mathcal{I}}}), \bot), \bot, \cdots, \bot) & \textit{if } \mathcal{C}(x_{\mathcal{I}}) \neq \bot\ , 1 \in \mathcal{I} \textit{ and } \boldsymbol{y}_{\mathcal{I}} = \bot, \textit{ where} \\ & \boldsymbol{y}_{\mathcal{I}} \overset{\text{def}}{=} (\mathcal{C}(\boldsymbol{x}_{\mathcal{I}}, f_I(\mathcal{C}(\boldsymbol{x}_{\mathcal{I}}), \boldsymbol{x}_{\bar{\mathcal{I}}})) \\ (\mathcal{C}(\boldsymbol{x}_{\mathcal{I}}, f_I(\mathcal{C}(\boldsymbol{x}_{\mathcal{I}}), \boldsymbol{x}_{\bar{\mathcal{I}}})), f_{\bar{I}}(\mathcal{C}(\boldsymbol{x}_{\mathcal{I}}), \boldsymbol{x}_{\bar{\mathcal{I}}})) & \textit{otherwise.} \end{array}$$

where $\bar{I} = [n] \setminus I$.

The first equation represents the case where the adversary makes some dishonest party to abort before invoking the trusted party. The second equation represents the case where the trusted party is invoked with possibly substituted inputs $C(\boldsymbol{x}_I)$ and is halted right after supplying the adversary with the I-part of the output $\boldsymbol{y}_I = f_I(\mathcal{C}(\boldsymbol{x}_{\mathcal{I}}), \boldsymbol{x}_{\bar{\mathcal{I}}})$. This case is allowed only when $1 \in \mathcal{I}$, i.e, the party P_1 can only be blamed for early abort. Finally, the third equation presents the case where the trusted party is invoked with possibly substituted inputs $C(\boldsymbol{x}_I)$, but is also allowed to answer to all the parties.

Definition 5 (Malicious adversaries–the real model). *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a n-ary functionality as defined above. Let $\boldsymbol{\Pi}$ be a protocol for computing f. The joint execution under $(\mathcal{I}, \mathcal{C})$ in the real model (on input sequence $\boldsymbol{x} = (x_1, \cdots, x_n)$), denoted by $\mathsf{REAL}_{\boldsymbol{\Pi},(\mathcal{I},\mathcal{C})}(\boldsymbol{x})$ is defined as the output sequence resulting of the interaction between the n parties where the messages of parties in I are computed according to $\mathcal{C}$ and the messages of parties not in I are computed according to $\boldsymbol{\Pi}$.*

Now that the ideal and real models are defined, we put forward the notion of security for a multi-party protocol. Informally, it says that a secure multi-party protocol in the real model emulates the ideal model.

Definition 6 (Security in the Malicious model). *Let f and $\boldsymbol{\Pi}$ be as in Def. 5. Protocol $\boldsymbol{\Pi}$ is said to* securely compute *f if there exists a polynomial-time computable transformation of polynomial-size circuit families $\mathcal{A} = \{\mathcal{A}_\lambda\}$ for the real model (of* Def. 5*) into polynomial-size circuit families $\mathcal{B} = \{\mathcal{B}_\lambda\}$ for the ideal model (of* Def. 4*) such that for every subset $I \subset [n]$ we have that $\{\mathsf{IDEAL}_{f,(\mathcal{I},\mathcal{B})}(\boldsymbol{x})\}_{\lambda \in \mathbb{N}, \boldsymbol{x} \in (\{0,1\}^\lambda)^n} \equiv_c \{\mathsf{REAL}_{\boldsymbol{\Pi},(\mathcal{I},\mathcal{A})}(\boldsymbol{x})\}_{\lambda \in \mathbb{N}, \boldsymbol{x} \in (\{0,1\}^\lambda)^n}$.*

3 Definition of Cryptographic Reverse Firewalls for Actively Secure MPCs

In this section, we present the definitions of reverse firewalls for activel;y secure MPC protocols, generalizing the definitions of [25, 12]. As in [25], we assume that a cryptographic protocol comes with some functionality (i.e., correctness) requirements $\mathcal{F}$ and some security requirements $\mathcal{S}$. For a party P and reverse firewall $\mathcal{W}$ we define $\mathcal{W} \circ P$ as the "composed" party in which the incoming and outgoing messages of A are "sanitized" by $\mathcal{W}$. In other words, $\mathcal{W}$ is applied to (1) the outgoing messages of P before they leave the local network of P and (2) the incoming messages of P before P sees them. We stress that the reverse firewall $\mathcal{W}$ neither shares any private input with party P nor does it get to know the output of party P. The firewall $\mathcal{W}$ is allowed to see only the public parameters of the system. Besides this, it can internally toss its own random coins and can also maintain state.

Let a MPC protocol Π run between the parties $\{P_1, \cdots, P_n\}$ with the set of honest parties denoted by H. We require that each honest party $\{P_i\}_{i \in H}$ is equipped with a reverse firewall $\mathcal{W}_i$. The first condition we want to the firewalls to satisfy is: The firewalls $\mathcal{W}_i$ should *preserve* the functionality of the MPC protocol, i.e., the composed parties $\mathcal{W}_i \circ P_i$ (for all $i \in H$) should not break the correctness of the protocol Π. Following [25, 12] we actually require the stronger property that the reverse firewalls be "stackable", i.e, many firewalls can be composed in series $\mathcal{W}_i \circ \cdots \circ \mathcal{W}_i \circ P_i$ without breaking the functionality of the protocol. In addition, we would want the firewalls $\mathcal{W}_i$ to preserve the security $\mathcal{S}$ of the underlying (actively secure) MPC protocol, even in the face of compromise of the honest parties. Our strongest notion of security requires that the security of the underlying MPC protocol be preserved even when all the honest parties P_i (of the underlying protocol) are *arbitrarily* corrupted (denote as $\overline{P_i}$). A weaker notion of security requires the security of the MPC protocol to hold, even when the honest parties P_i are tampered in a *functionality-maintaining* way (denoted by $\widehat{P_i}$), i.e., when the tampered implementation still maintains the functionality $\mathcal{F}$ of the protocol[4]. We write $\Pi_{\{P_i \to \widehat{P_i}\}_{i \in H}}$ to represent the protocol in which the role of the honest parties $\{P_i\}_{i \in H}$ are replaced by parties $\{\widehat{P_i}\}_{i \in H}$. Further, we require *exfiltration resistance* from the reverse firewalls, which informally says that "no corrupt implementation of a party P can leak any information through the firewall". Finally, following [12], we will also need the notion of "detectable failure" from the reverse firewall. Informally, this notion stipulates that a protocol fails detectably if we can distinguish transcripts of valid runs of a protocol from invalid transcripts. This property will be used by the firewall of a large protocol to test whether some sub-protocol failed or not. We now formally define all these properties below.

Definition 7. (Functionality-maintaining CRF). *For any reverse firewall $\mathcal{W}$ and a party P, let $\mathcal{W}^1 \circ P = \mathcal{W} \circ P$, and $\mathcal{W}^k \circ P = \underbrace{\mathcal{W} \circ \cdots \circ \mathcal{W}}_{k \text{ times}} \circ P$. A reverse firewall $\mathcal{W}$ maintains functionality $\mathcal{F}$ for a party P in protocol Π if Π satisfies $\mathcal{F}$, the protocol $\Pi_{P \to \mathcal{W} \circ P}$ satisfies $\mathcal{F}$, and the protocol $\Pi_{P \to \mathcal{W}^k \circ P}$ also satisfies $\mathcal{F}$.*

Definition 8. (Security-preserving CRF for Maliciously secure MPCs). *Let $\boldsymbol{\Pi}$ be a multi-party protocol run between the parties $P_1, \cdots, P_n$ satisfying functionality requirement $\mathcal{F}$ and is secure against static malicious adversaries (see Def. 6). Let*

[4] In this work, we assume that the functionality-maintaining corruptions never break the functionality of the protocol. A stronger requirement, as proposed in [25], requires that the functionality-maintaining corruptions may behave arbitrarily a negligible fraction of time. While we prove our constructions to be secure according to our notion, it is possible to modify our constructions easily to meet the requirements of [25]. In order to avoid complications, we choose not to consider the above case, and will present this in an extended version of our work.

$\mathcal{I} = \{i_1, \cdots, i_q\} \subset [n]$ be the indices of the parties that are corrupt (in the sense of "standard" corruption) by the adversary $\mathcal{A}$ in the underlying protocol $\boldsymbol{\Pi}$. Also, let $= \{\overline{P_i}\}_{i\in\overline{\mathcal{I}}}$ (resp., $\{\widehat{P}_i\}_{i\in\overline{\mathcal{I}}}$) denote the set of arbitrary (resp. "functionality-maintaining") tampered implementations of the honest parties (where, $\overline{\mathcal{I}} = [n] \setminus \mathcal{I}$). We assume that each honest party $\{P_i\}_{i\in\overline{\mathcal{I}}}$ is equipped with its corresponding reverse firewall $\{\mathcal{W}_i\}_{i\in\overline{\mathcal{I}}}$. Denote by $\{P_i^* \leftarrow \mathcal{W}_i \circ \overline{P_i}\}_{i\in\overline{\mathcal{I}}}$ the set of composed parties comprising of the tampered implementations $\{\overline{P_i}\}_{i\in\overline{\mathcal{I}}}$ (of parties $\{P_i\}_{i\in\overline{\mathcal{I}}}$) of the honest parties composed with their respective firewalls $\{\mathcal{W}_i\}_{i\in\overline{\mathcal{I}}}$. Then:

- *We say that the reverse firewalls $\mathcal{W}_i$ for parties $\{P_i\}_{i\in\overline{\mathcal{I}}}$ strongly preserves security of the protocol $\boldsymbol{\Pi}$, if there exists a polynomial-time computable transformation of polynomial-size circuit families $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ for the real model (see* Def. 5*) into polynomial-size circuit families $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ for the ideal model (see* Def. 4*) such that for every subset $I \subset [n]$, every input sequence $\boldsymbol{x} = (x_1, \cdots, x_n) \in (\{0,1\}^\lambda)^n$, and every tampered implementation $\{\overline{P_i}\}_{i\in\overline{\mathcal{I}}}$ we have that:*

$$\{\mathsf{IDEAL}_{f,(\mathcal{I},\mathcal{B})}(\boldsymbol{x})\}_{\lambda\in\mathbb{N},\boldsymbol{x}\in(\{0,1\}^\lambda)^n} \equiv_c \{\mathsf{REAL}_{\boldsymbol{\Pi}_{\{P_i\to P_i^*\}_{i\in\overline{I}}},(\mathcal{I},\mathcal{A})}(\boldsymbol{x})\}_{\lambda\in\mathbb{N},\boldsymbol{x}\in(\{0,1\}^\lambda)^n}.$$

- *We say that the reverse firewalls $\mathcal{W}_i$ for parties $\{P_i\}_{i\in\overline{I}}$ weakly preserves security of the protocol $\boldsymbol{\Pi}$, if the above condition holds, but only with respect to "functionality-maintaining" tampered implementations $\{\widehat{P}_i\}_{i\in\overline{I}}$ of parties $\{P_i\}_{i\in\overline{I}}$.*

We also need the notion of *exfiltration-resistance* from the reverse firewall. In formally, a reverse firewall is exfiltration-resistant if "no corrupt implementation of a party can leak any information through the firewall". Our definition of exfiltration-resistance generalizes the definition of [25, 12] in the multi-party setting.

Definition 9. (Exfiltration-resistant CRF). *Let Π be a multi-party protocol run between the parties $P_1, \cdots, P_n$ satisfying functionality $\mathcal{F}$ and having reverse firewalls $\mathcal{W}_i$ for the set of honest parties $\{P_i\}_{i\in\overline{\mathcal{I}}}$ (recall that $\mathcal{I}$ is the set of users corrupted by the adversary in the underlying protocol Π). Then $\forall i \in \overline{\mathcal{I}}$:*

- *We say that the firewall $\mathcal{W}_i$ is* strongly exfiltration-resistant for party P_i against all other parties $(P_1, \cdots, P_{i-1}, P_{i+1}, \cdots, P_n)$, *if for any PPT adversary $\mathcal{E}$, the advantage $\mathsf{Adv}^{\mathsf{LEAK}}_{\mathcal{E},\mathcal{W}_i}(\lambda)$ of $\mathcal{E}$ in the game* LEAK *as shown below is negligible in the security parameter λ,*
- *We say that the firewall $\mathcal{W}$ is* weakly exfiltration-resistant for party P_i against the other parties $(P_1, \cdots, P_{i-1}, P_{i+1}, \cdots, P_n)$, *if for any PPT adversary $\mathcal{E}$, the advantage $\mathsf{Adv}^{\mathsf{LEAK}}_{\mathcal{E},\mathcal{W}_i}(\lambda)$ of $\mathcal{E}$ in the game* LEAK *(see Figure 1)is negligible in the security parameter λ, provided that P_i maintains functionality $\mathcal{F}$ for P_i.*

The advantage of any adversary $\mathcal{E}$ in the game LEAK *is defined as:* $\mathsf{Adv}^{\mathsf{LEAK}}_{\mathcal{E},\mathcal{W}_i}(\lambda) = \left|\Pr[\mathsf{LEAK}(\Pi, i, \{P_1, \cdots, P_n\}, \mathcal{W}_i, \lambda) = 1] - \frac{1}{2}\right|$.

Finally, we define another technical condition related to *detectable failures* of reverse firewalls, as presented in [12]. First, we recall the definition for what it means for a transcript to be *valid*, and then define detectable failures.

```
Proc. $\underline{\mathsf{LEAK}(\mathit{\Pi}, i, \{P_1, \cdots, P_n\}, \mathcal{W}_i, \lambda)}$
    $(\overline{P_1}, \cdots, \overline{P_n}, I) \leftarrow \mathcal{E}(1^\lambda)$.
    $b \xleftarrow{\$} \{0,1\}$.
    IF $b = 1$, $P_i^* \leftarrow \mathcal{W}_i \circ \overline{P_i}$.
    ELSE, $P_i^* \leftarrow \mathcal{W}_i \circ P_i$.
    $\mathcal{T}^* \leftarrow \mathit{\Pi}_{P_i \to P_i^*, \{P_j \to \overline{P_j}\}_{j \in [n \setminus i]}}(I)$.
    $b^* \leftarrow \mathcal{E}(\mathcal{T}^*, \{\mathsf{st}_{P_j}\}_{j \in [n \setminus i]})$.
    OUTPUT $(b = b^*)$
```

Fig. 1. $\mathsf{LEAK}(\mathit{\Pi}, i, \{P_1, \cdots, P_n\}, \mathcal{W}_i, \lambda)$ is the exfiltration-resistance security game for a reverse firewall $\mathcal{W}_i$ for a party P_i in protocol $\mathit{\Pi}$ against the set of parties $\{P_j\}_{j \in [n \setminus i]}$ with input I. $\mathcal{E}$ is the adversary, λ is the security parameter, $\{\mathsf{st}_{P_j}\}_{j \in [n \setminus i]}$ denote the states of the parties $\{P_j\}_{j \in [n \setminus i]}$ after the run of the protocol, I is the valid input for $\mathit{\Pi}$, and $\mathcal{T}^*$ is the transcript of running the protocol $\mathit{\Pi}_{P_i \to P_i^*, \{P_j \to \overline{P_j}\}_{j \in [n \setminus i]}}(I)$.

Definition 10 (Valid Transcripts [12]). *A sequence of bits r and private input I generate transcript $\mathcal{T}$ in protocol $\mathit{\Pi}$ if a run of the protocol $\mathit{\Pi}$ with input I in which the parties' coin flips are taken from r results in the transcript $\mathcal{T}$. A transcript $\mathcal{T}$ is a* valid transcript *for protocol $\mathit{\Pi}$ if there is a sequence r and private input I generating $\mathcal{T}$ such that no party outputs $\perp$ at the end of the run. A protocol has* unambiguous transcripts *if for any valid transcript $\mathcal{T}$, there is no possible input I and coins r generating $\mathcal{T}$ that results in a party outputting $\perp$.*

Definition 11 (Detectable failure). *A reverse firewall $\mathcal{W}$* detects failure *for party P in protocol $\mathit{\Pi}$ if (a) $\mathit{\Pi}_{P \to \mathcal{W} \circ P}$ has unambiguous transcripts; (b) the firewall outputs a special symbol $\perp$ when run on any transcript that is not valid for $\mathit{\Pi}_{P \to \mathcal{W} \circ P}$, and (c) there is a polynomial-time deterministic algorithm that decides whether a transcript $\mathcal{T}$ is valid for $\mathit{\Pi}_{P \to \mathcal{W} \circ P}$.*

4 Reverse Firewalls and Actively secure MPCs

In this section, we discuss the relationship between actively secure MPC protocols and reverse firewalls. In this work, we consider *computationally-secure* MPC protocols. For the protocol to be secure, we need to assume that atleast one of the parties participating in the MPC protocol is "honest". However, in the setting of reverse firewalls, this assumption may not hold true, and in general, we cannot rely on trusted implementation of any of the parties to guarantee security of the resulting MPC protocol. In particular, in this setting, one may consider a scenario where all the parties may be arbitrarily corrupted. To provide any sort of meaningful security guarantees in such a strong corruption model, we assume that each of the *honest* parties participating in the MPC protocol are equipped with a cryptographic reverse firewall. As mentioned earlier, none of the firewalls share any secrets with any of the parties, nor can it access the outputs of the corresponding parties. The firewall has access to only the public parameters used in the protocol. All the incoming and outgoing messages sent and received by the parties are modified by the firewall. The hope is that: even if the honest parties are corrupted, the firewall can somehow sanitize the outgoing and incoming messages in such a way that the security of the original MPC protocol (where there is at least one honest party) is preserved.

Ideally, we would like to build reverse firewalls for the MPC protocol, where all the *honest* parties can be arbitrarily corrupted. However, in order to accomplish this goal, we will need to consider the following scenario: Suppose that one of the parties which was assumed to be honest in the original MPC protocol refuses to communicate (also called "attack by refusal" in [12]) in this new model of corruption. To guarantee security against this attack, clearly the firewall needs to produce a message which looks indistinguishable from the message the honest party would have sent in the original MPC protocol. In order words, the firewall needs to *simulate* the behavior of this (honest) party in our new corruption model, where the same party can be arbitrarily corrupted. Now suppose that, the party has a public-secret key pair and it uses the secret key to compute some message at some point in the protocol (say, a signature on the transcript so far). Clearly, this action *cannot* be simulated by the firewall, since it does not have access to the secret key of the party. Hence, in this setting, where the parties have access to key pairs (which will indeed be the case for us), achieving security against strong or arbitrary corruption is *impossible.*

To circumvent the above impossibility result, we consider a *hybrid* model of corruption, which is slightly weaker than the corruption model mentioned above. In particular, in our model, up to $n-1$ parties can be arbitrarily corrupted, where n is the total number of parties participating in the protocol. The remaining honest parties can also be corrupted, albeit, in a *functionality-maintaining* way. In a functionality-maintaining tampered implementation of a party, the adversary may deviate arbitrarily from the protocol, as long as it does not break its functionality. Intuitively, this models "more conspicuous" adversaries whose tampered circuit(s) will be noticed by honest parties participating in the protocol with non-negligible probability [25].

5 Dolev-Strong Broadcast for Reverse Firewalls

As mentioned earlier, we will assume the availability of a broadcast channel for our construction of the actively-secure MPC protocol in the reverse firewall (CRF) setting. However, in the CRF setting, the assumption of broadcast channels may be stronger than the classical setting. To this end, we present a compiler for reverse firewalls for the broadcast model. We instantiate the broadcast protocol using a version of the classical Dolev-Strong protocol [14], secure in the CRF setting. The protocol of [14] shows that one can simulate a broadcast channel using public-key infrastructure, in particular using signature schemes as the authentication mechanism. In our construction, we replace the signature scheme from [14] with *unique* signatures. Intuitively this works since: on any input in the Dolev-Strong protocol, the only allowed message consists of adding a signature on a well-defined message. The signature is either sent or added to a valid set. Since the signatures are unique, this leaves only one possible message that a (even corrupted) party can send. The latter holds since we assume that the parties are corrupted in a functionality-maintaining way.

Ports: For each party P_i there is a protocol port Cast_i. There are also special ports Leak, $\mathsf{Replace}$, and $\mathsf{Deliver}$ connected to the adversary.

Syntax: The inputs on Cast_i from parties P_i can be of the form $(\mathsf{bid}, \mathsf{P}_i, m)$, where bid is a broadcast ID and m is a message. Furthermore, parties $\mathsf{P}_j \neq \mathsf{P}_i$ can give inputs of the form $(\mathsf{bid}, \mathsf{P}_i, ?)$ indicating that they know that P_i is about to give an input. Parties P_j can give outputs of the form $(\mathsf{bid}, \mathsf{P}_i, m)$ indicating that they think P_i has broadcast m.

Input: On input $(\mathsf{bid}, \mathsf{P}_i, m)$ on Cast_i the IF outputs $(\mathsf{bid}, \mathsf{P}_i, m)$ on Leak and stores $(\mathsf{bid}, \mathsf{P}_i, m)$.

Replace: On input $(\mathsf{bid}, \mathsf{P}_i, m)$ on $\mathsf{Replace}$, where P_i is corrupted the IF stores $(\mathsf{bid}, \mathsf{P}_i, m)$, overwriting any previous value of form $(\mathsf{bid}, \mathsf{P}_i, \cdot)$.

User Contract: Honest parties must give input according to the following contract.

- **Synchrony:** If in some round any honest party gives an input of the form $(\mathsf{bid}, \mathsf{P}_i, \cdot)$, then in that round all honest parties give an input of the form $(\mathsf{bid}, \mathsf{P}_i, \cdot)$.
- **Unique identifiers:** If an honest party is honest then for each bid it gives at most one input of the form $(\mathsf{bid}, \cdot, \cdot)$.

Total Breakdown: On input $((\mathsf{bid}, \mathsf{P}_i, m), \mathsf{P}_j)$ on $\mathsf{Deliver}$ at a point in time after the user contract was broken, output $(\mathsf{bid}, \mathsf{P}_i, m)$ to P_j.

Ideal Functionality Contract: If the inputs of the honest parties are according to the user contract, then the IF gives outputs according to the following rule.

- **Output:** If an input of the form $(\mathsf{bid}, \mathsf{P}_i, \cdot)$ was input on Cast_k for an honest P_k in round r, then in round $r + n + 2$ rounds, find the stored value $(\mathsf{bid}, \mathsf{P}_i, m)$. If none is stored, use $m = \mathsf{NoMsg}$. Then output $(\mathsf{bid}, \mathsf{P}_i, m)$ on Cast_j for all honest P_j.

Fig. 2. An ideal functionality Cast for Broadcast.

Here we present a version of the Dolev-Strong protocol suitable for reverse firewalls. The setup will be that each party gets a signing key. We phrase the protocols and ideal functionalities in terms of sending values on ports. Send a message m on a port Port and a party P_i just means sending (Port, m) to P_i. The protocol implements the ideal functionality in Fig. 2. This IF will broadcast a value in $n + 2$ rounds. It requires that all honest users start the protocol in the same round. We model this by allowing the IF to behave arbitrarily when this user contract is broken. This makes it trivial to simulate the IF when the user contract is broken: simply ask the IF to output the same (faulty) messages as the protocol. Hence in the proof of security we can focus on just the case where the user contract is never broken.

The protocol is given in Fig. 3. The analysis of Dolev-Strong is standard by now, but we sketch the proof for completeness. Assume that some honest party output $(\mathsf{bid}, \mathsf{P}_i, \cdot)$ in round r. Then by the user contract all honest parties got input $(\mathsf{bid}, \mathsf{P}_i, \cdot)$ in round $r - n - 2$. Therefore they all output $(\mathsf{bid}, \mathsf{P}_i, \cdot)$ in round r. To see that they output the same m we show that $\mathsf{Relayed}_i(\mathsf{bid}, m) = \mathsf{Relayed}_j(\mathsf{bid}, m)$ for all honest P_i and P_j when they give output. Namely, if $\mathsf{Relayed}_i(\mathsf{bid}, m) = \top$ then it was set to this after seeing a valid signature set in some round r after input $(\mathsf{bid}, \mathsf{P}_i, \cdot)$. Let I be the round where $(\mathsf{bid}, \mathsf{P}_i, \cdot)$ was input. The valid set had size $r - 1$ and was received in round $I + r$. It had $r - 1$ signatures from distinct parties and none from P_i, so it had size at most $n - 1$. So $r \leq n$. Then P_i added its signature and relayed the set. Therefore it is received as a valid set in the next round by all parties P_j who did not already set $\mathsf{Relayed}_j(\mathsf{bid}, m) = \top$. This is round at most $I + n + 1$. Therefore all honest parties P_j will set $\mathsf{Relayed}_j(\mathsf{bid}, m) = \top$

Initialize Party P_i learns the private signing key sk_i and the public keys of all parties, $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$. Then it initializes a map $\mathsf{Relayed}_i$ which is $\mathsf{Relayed}_i(\mathsf{bid}, m) = \bot$ for all possible broadcast identifiers bid and messages m.

Broadcast On input $(\mathsf{bid}, \mathsf{P}_i, m)$ on Cast_i compute $\sigma_i \leftarrow \mathsf{Sig}_{\mathsf{sk}_i}(\mathsf{bid}, m)$, $\mathsf{SigSet} = \{\sigma_i\}$, set $\mathsf{Relayed}_i(\mathsf{bid}, m) = \top$, and send $(\mathsf{bid}, \mathsf{P}_i, m, \mathsf{SigSet})$ to all parties.

Relay In round r after input $(\mathsf{bid}, \mathsf{P}_i, ?)$, if $\mathsf{P}_j \neq \mathsf{P}_i$ receives a message of form $(\mathsf{bid}, m, \mathsf{SigSet})$, where SigSet is a set of signatures, and if $\mathsf{Relayed}_i(m) = \bot$, proceed as follows. Call SigSet valid for $(\mathsf{bid}, \mathsf{P}_i, m)$ in round r if it contains signatures σ_k from exactly $r - 1$ distinct parties P_k such that $\mathsf{Ver}_{\mathsf{vk}_k}(\mathsf{bid}, m, \sigma_k) = \top$. Furthermore, one of these parties has to be P_i and none of them are P_j. If SigSet is valid, then compute $\sigma_j \leftarrow \mathsf{Sig}_{\mathsf{sk}_j}(\mathsf{bid}, m)$, let $\mathsf{SigSet}' \leftarrow \mathsf{SigSet} \cup \{\sigma_j\}$ and send $(\mathsf{bid}, m, \mathsf{SigSet}')$ to all parties. Then set $\mathsf{Relayed}_i(\mathsf{bid}, m) = \top$.

Output In round $n+2$ after input $(\mathsf{bid}, \mathsf{P}_i, ?)$, party P_j computes its output as follows. If there is exactly one message m such that $\mathsf{Relayed}_i(\mathsf{bid}, m) = \top$, then output $(\mathsf{bid}, \mathsf{P}_i, m)$ on Cast_j. Otherwise, output $(\mathsf{bid}, \mathsf{P}_i, \mathsf{NoMsg})$ on Cast_j.

Fig. 3. The Dolev-Strong Protocol, $\mathsf{DolevStrong}$

before round $I + n + 2$. The honest parties give output in round $I + n + 2$. Hence $\mathsf{Relayed}_i(\mathsf{bid}, m) = \mathsf{Relayed}_j(\mathsf{bid}, m)$ for all honest P_i and P_j when they give output.

In Fig. 4 we present a wrapper for party P_i in Dolev-Strong with unique signatures. We will only treat this case here. The case with randomizable signatures follows by letting the wrapper randomize the outgoing signatures.

Initialize The wrapper for party P_i learns the public keys of all parties, $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$. Then it initializes a map $\mathsf{Relayed}_i$ which is $\mathsf{Relayed}_i(\mathsf{bid}, m) = \bot$ for all possible broadcast identifiers bid and messages m.

Broadcast On input $(\mathsf{bid}, \mathsf{P}_i, m)$ on Cast_i it expects P_i in the same round to send $(\mathsf{bid}, \mathsf{P}_i, m, \mathsf{SigSet})$, where $\mathsf{SigSet} = \{\sigma_i\}$ and $\mathsf{Ver}_{\mathsf{vk}_i}(\mathsf{bid}, m, \sigma_i) = \top$. If so, then it lets $(\mathsf{bid}, \mathsf{P}_i, m, \mathsf{SigSet})$ pass. Otherwise, output Refusal.

Relay In round r after input $(\mathsf{bid}, \mathsf{P}_i, ?)$, if $\mathsf{P}_j \neq \mathsf{P}_i$ receives a message of form $(\mathsf{bid}, m, \mathsf{SigSet})$, where SigSet is a set of signatures, and if $\mathsf{Relayed}_i(m) = \bot$, proceed as follows. Call SigSet valid for $(\mathsf{bid}, \mathsf{P}_i, m)$ in round r if it contains signatures σ_k from exactly $r - 1$ distinct parties P_k such that $\mathsf{Ver}_{\mathsf{vk}_k}(\mathsf{bid}, m, \sigma_k) = \top$. Furthermore, one of these parties has to be P_i and none of them are P_j. If SigSet is valid, then expect P_j in the same round to send $\mathsf{SigSet}' = \mathsf{SigSet} \cup \{\sigma_j\}$ where $\mathsf{Ver}_{\mathsf{vk}_j}(\mathsf{bid}, m, \sigma_j) = \top$. If so, let the message pass. Otherwise, output Refusal.

Output In round $n+2$ after input $(\mathsf{bid}, \mathsf{P}_i, ?)$, the allowed output of P_j is computed as follows. If there is exactly one message m such that $\mathsf{Relayed}_i(\mathsf{bid}, m) = \top$, then $(\mathsf{bid}, \mathsf{P}_i, m)$ is expected. Otherwise, $(\mathsf{bid}, \mathsf{P}_i, \mathsf{NoMsg})$ is expected. If P_j does not give the expected output on Cast_j, output Refusal. If it gives another output, then drop it and output Deviation.

Deviation If P_i outputs any value not explicitly expected in one of the above rules, drop it and output Deviation.

Fig. 4. The Dolev-Strong Wrapper Wrap for the case with unique signatures

For two programs A and B we use $A \equiv B$ to mean that they give the same outputs on the same input sequences. The following lemma follows by construction.

Lemma 1. *The wrapper* Wrap *is input-output preserving, i.e.,* $\mathsf{Wrap}(\mathsf{P}_i) \equiv \mathsf{P}_i$*.*

Let P'_i be a possibly corrupt implementation of P_i. We call it deviating if it makes $\mathsf{Wrap}(\mathsf{P}'_i)$ output DEVIATION. We call it refusing if it makes $\mathsf{Wrap}(\mathsf{P}'_i)$ output REFUSAL. We call it covert if it does not make $\mathsf{Wrap}(\mathsf{P}'_i)$ output neither DEVIATION not REFUSAL.

The following lemma is also straight forward.

Lemma 2. *Assume that* $\mathsf{DolevStrong}$ *is implemented using unique signatures. Let* P'_i *be a possibly corrupt implementation of* P_i*. If* P'_i *is covert, then on any input sequence to* $\mathsf{Wrap}(\mathsf{P}'_i)$ *there is exactly one message that* P'_i *can send, and it will always send this message.*

Namely, on any input in the Dolev-Strong protocol, the only allowed message consists of adding a signature on a well-defined message. This signature is either sent or added to a valid set. The message is known by the wrapper. Since signatures are unique, this leaves one possible message P'_i can send. It *will* send it, as it is not refusing. It will not send other messages, as it is not deviating.

Putting the above together we get the following theorem.

Theorem 2. *Assume that* $\mathsf{DolevStrong}$ *is implemented using unique signatures. Let* P'_i *be a possibly corrupt implementation of* P_i*. If* P'_i *is covert, then* $\mathsf{Wrap}(\mathsf{P}'_i) \equiv \mathsf{P}_i$*.*

Namely, by the second lemma we have that $\mathsf{Wrap}(\mathsf{P}'_i) \equiv \mathsf{Wrap}(\mathsf{P}_i)$. By the first lemma we have that $\mathsf{Wrap}(\mathsf{P}_i) \equiv \mathsf{P}_i$. And $\equiv$ is transitive.

5.1 A Compiler for Reverse Firewalls for the Broadcast Model

We now describe how to take a reverse firewall R for a protocol π for the Cast-hybrid model and make it into a reverse firewall R' for the protocol $\pi' = \pi[\mathsf{DolevStrong}/\mathsf{Cast}]$ which uses Dolev-Strong for broadcast. Let Q_i be the i'th party in π. Let P_i be the i party in $\mathsf{DolevStrong}$. Let R_i be the i'th party in π'. Then R_i consists of Q_i and P_i. The reverse firewall R' is composed of R and Wrap. We apply the reverse firewall R' to R_i as follows. Apply R to Q_i. Whenever R lets Q_i broadcast a message $(\mathsf{bid}, \mathsf{P}_i, m)$, input it to $\mathsf{Wrap}(\mathsf{P}_i)$ on Cast_i. Then send only message that are allowed by $\mathsf{Wrap}(\mathsf{P}_i)$ and send all in-coming trafic to $\mathsf{Wrap}(\mathsf{P}_i)$. If $\mathsf{Wrap}(\mathsf{P}_i)$ outputs $(\mathsf{bid}, \mathsf{P}_i, m)$ on Cast_j then give it to $R(\mathsf{Q}_i)$.

6 Actively secure MPC protocols using Reverse Firewalls

In this section, we present a construction of multi-party computation (MPC) protocol secure against malicious adversaries in the setting of reverse firewalls. As mentioned above, we only consider computationally-secure MPC protocols. The starting point of our construction is the actively-secure MPC protocol of Goldreich, Micali and Wigderson

[18, 17] (henceforth referred to as the GMW protocol). Their methodology works by first presenting a MPC protocol secure against semi-honest adversaries, and then compiling it into a protocol secure against malicious adversaries. The resulting actively secure GMW protocol can tolerate a corruption of up to $n-1$ parties, where n is the number of parties participating in the protocol. We begin with an informal exposition of the GMW compiler.

Informal description of the GMW compiler. As mentioned before, the GMW protocol [18, 17] first constructs a semi-honest MPC protocol, and then compiles it to one which is secure against malicious adversaries. Recall that, in the semi-honest protocol all the parties follow the protocol specification exactly. However, in the malicious model, the parties may deviate arbitrarily from the protocol. The way that the GMW protocol achieves security against malicious adversaries is by somehow enforcing the parties to behave in a semi-honest manner. However, this only makes sense relative to a given input and a random tape. The GMW protocol achieves this in the following way:

- All the parties first commit to their inputs by running a *multi-party input commitment* protocol. Note that, before the protocol starts each party may replace their given inputs with arbitrary bit strings. However, the security of this protocol guarantees that, once they commit to their inputs, it cannot be changed afterwards during the course of execution of the protocol.
- The parties run an actively-secure *multi-party (augmented) coin tossing* protocol to fix their random tapes (to be used in the actual MPC protocol). This protocol ensures that all the parties have a uniformly random tape.
- After these first two steps, each party holds its own uniformly random tape, and the commitments to other party's inputs and random tapes. Hence, the parties can now be forced to behave properly in the following way: the view of each party in the MPC protocol is simply a deterministic function of its own input, random tape and the (public/broadcast) messages received so far in the protocol. Hence, when a party sends a new message it also proves in zero-knowledge that the computation was correctly done, as per the protocol specification. The soundness of the proof system guarantees that even a malicious adversary cannot deviate from the protocol, while the zero-knowledge property ensures that nothing other than the validity of each computational step is revealed to the adversary. This phase is also called the *protocol emulation* phase.

When we consider the actively-secure GMW protocol in the reverse firewall settings, we must ensure that the above-mentioned protocols remain functional and secure in the setting of reverse firewalls. Hence, we need to design reverse firewalls for each of the three main protocols (as discussed above) used in the GMW compiler. Finally, to enable the working of the compiler, we need to show that the reverse firewalls for each of these protocols compose together. To this end, we first propose a multi-party augmented coin-tossing protocol with *reduced round-complexity* (see section 6.1) by appropriately extending the two-party coin-tossing protocol of Lindell [23]. We then present a reverse firewall for this multi-party coin-tossing protocol in section 6.2. In sections 6.3 and 6.4, we present reverse firewalls for the multi-party input commitment and the multi-party authenticated computation protocols.

6.1 Multi-party Augmented Coin-Tossing into the Well

The multi-party augmented coin tossing protocol is used to generate *random pads* for all the parties participating in an actively secure multi-party computation protocol. Each party obtains the bits of the random-pad to be held by it, whereas the other parties obtains commitments to these bits. These random pads serve as the random coins of the corresponding parties to emulate the semi-honest MPC protocol. Intuitively, this multi-party coin-tossing functionality guarantees that, at the end of this protocol the malicious parties can either abort or they end up with a uniformly distributed random pad. However, the original coin-tossing protocol of GMW [18, 17] was rather inefficient in terms of round complexity. This is because the protocol of [18, 17] required polynomially many rounds to generate a polynomially long random pad, since single coins were tossed sequentially in each round. Later, Lindell [23] showed a constant round two-party protocol for augmented parallel coin-tossing into the well using a "commit-and-proof" framework. In Fig. 5, we extend the protocol of [23] in the multi-party setting with round-complexity *only* 3[5] and achieving a comparable level of security as in [23]. In section 6.2, we present a reverse firewall for our multiparty augmented coin-tossing protocol.

In our case, we slightly *modify* the augmented coin-tossing functionality [18, 17, 23] to fit the framework of reverse firewalls. In particular, in the traditional augmented coin-tossing functionality, the initiating party receives a tuple (x, r) and all other parties receive a commitment $\mathsf{com}(x; r)$ of the same tuple, for a statistically/perfectly hiding (and computationally binding) commitment scheme. Looking ahead, the value x would serve as the random coins of the initiating party in the protocol. For our purpose, we change the above functionality as follows: The initiating party receives a tuple (x, r) and all other parties receive a commitment $\mathsf{com}(x; r')$ for a potentially different r'. This means that, all the other parties still receive a commitment of the same random coin x but potentially under different randomness. Intuitively, this is needed for the following reason: When the initiating party is corrupted in a functionality-maintaining way, it might sample the randomness r used in the commitment $\mathsf{com}(x; r)$ in a malicious way. This might act as a channel to exfiltrate sensitive information to other parties. Hence, the firewall (of the initiating party) should re-randomize the commitment string to thwart this attack vector.

For our firewalled augmented coin-tossing into the well protocol, we additionally need the commitment scheme com to be *additively homomorphic* (other than being *statistically/perfectly hiding* (and computationally binding)), and also require the NIZK

[5] Although the protocol of Lindell [23] is constant round, its round-complexity is greater than 4 due to the use of (constant-round) zero-knowledge proofs. We use NIZK arguments in a natural way to shrink the round-complexity of the protocol to 3, albeit introducing a trusted setup assumption, as required for NIZK protocols.

argument system to be *controlled-malleable* simulation-sound extractable with respect to the above homomorphic operation.

Definition 12 (Multi-party Augmented Parallel Coin-Tossing into the Well). *An n-party* augmented coin-tossing into the well *protocol is an n-party protocol for securely computing the following functionality with respect to a fixed commitment scheme $\{\mathcal{G}_\lambda, K_\lambda, \mathsf{com}_\lambda\}_{\lambda\in\mathbb{N}}$,*

$$(1^\lambda, \cdots, 1^\lambda) \to \big((U_t, U_{t\cdot\lambda}), \mathsf{com}_\lambda(U_t; U'_{t\cdot\lambda}), \cdots, \mathsf{com}_\lambda(U_t; U'_{t\cdot\lambda})\big) \tag{1}$$

where U_m denotes the uniform distribution over m-bit strings, and we assume that com *requires λ random bits to commit to each bit. Note that the string $U'_{t\cdot\lambda}$ maybe different from the string $U_{t\cdot\lambda}$.*

Thus, all the parties other than the one who initiates the protocol receive a commitment to a uniformly random t bit string, and the committing/initiating party receives the random string and a decommitment to it. In the final compiler, the t bit strings will be used as random pads for the parties and the decommitment value is used to provide consistency checks for each step of the protocol (via (non-interactive) zero-knowledge proof).

W.l.o.g, we denote some party P_i ($i \in [n]$) to be the initializing party in the protocol below (see Fig. 5), and all the other parties P_j (where $j \in [n] \setminus i$) receive a commitment to the random string of P_i. In the final MPC protocol, each party will need to run an independent instance of the multi-party coin-tossing protocol shown below.

Let $\{\mathcal{G}_\lambda, K_\lambda, \mathsf{com}_\lambda\}_{\lambda\in\mathbb{N}}$ be a *statistically/perfectly hiding* and *computationally binding* commitment scheme. Also, let $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V})$ be a *strong simulation-extractable non-interactive zero-knowledge* (SSE-NIZK) argument system for the following language: $\mathcal{L} = \{c, (x, y) \mid c = \mathsf{com}_\lambda(x; y)\}$.

Inputs: Each party gets as input the security parameter 1^λ.
Convention: As mentioned above, we denote the initializing party in each round by party P_i. Any deviation from the protocol, by a party other than Party P_i, will be interpreted as a canonical legitimate message. In case P_i aborts or is detected cheating, all honest parties halt outputting the special symbol $\perp$.

(i) Party P_i chooses a random string $s_i \in_R \{0,1\}^m$. It then computes $c_i = \mathsf{com}_\lambda(s_i; r_i)$ for a random r_i using a computationally binding commitment scheme. P_i then computes a proof $\pi_i \leftarrow \mathsf{P}(\sigma_{crs}, c_i, (s_i, r_i))$ using the SSE-NIZK argument system. P_i then places the tuple (c_i, π_i) on the broadcast channel. In case the proof π_i does not verify with respect to c_i, all the parties abort with output $\perp$.

(ii) For $j \in [n] \setminus i$, party P_j selects $s_j \in_R \{0,1\}^m$ and places s_j on the broadcast channel.

(iii) Party P_i sets $s = s_i \oplus_{j\in[n]\setminus i} s_j$, and computes $y = \mathsf{com}_\lambda(s; r'_i)$, for a random r'_i. P_i then proves in zero-knowledge that there exists a tuple (s_i, r_i, r'_i) such that $c_i = \mathsf{com}_\lambda(s_i; r_i)$ and $y = \mathsf{com}_\lambda(s_i \oplus_{j\in[n]\setminus i} s_j; r'_i)$. It then places the tuple (y, π) on the broadcast channel. As before, if the proof π does not verify with respect to (c_i, y), all the parties abort with output $\perp$.

Outputs: Party P_i sets its local output to (s, r'_i) and all the other parties set their local output to be y, provided they did not halt with output $\perp$ before.

Fig. 5. Multi-party Augmented Parallel Coin-Tossing into the Well.

Theorem 3. *Let* $\{\mathcal{G}_\lambda, K_\lambda, \mathsf{com}_\lambda\}_{\lambda\in\mathbb{N}}$ *be a perfectly hiding and computationally binding commitment scheme. Also, let* $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V}, \mathsf{ZKEval})$ *be a strong simulation-extractable non-interactive zero-knowledge argument system for the language defined in* Fig. 5*. Then the protocol shown in* Fig. 5 *is a secure protocol for multi-party augmented coin-tossing into the well.*

The proof of this theorem is a straightforward generalization of the proof of the two-party coin-tossing protocol of [23] to the multi-party setting.

6.2 Multi-party Augmented Coin-Tossing using Reverse Firewalls.

In this section, we present a cryptographic reverse firewall (CRF) for the multi-party augmented parallel coin-tossing protocol, as shown in Fig. 6. We present a single reverse firewall $\mathcal{W}_1$ for this protocol that happens to work for all the honest parties. However, each of the honest parties involved in the coin-tossing protocol should be equipped with their own CRF. It so happens that the "code" of the firewall is the same for all these parties.

Main Idea. The main idea underlying the multi-party coin-tossing protocol from Fig. 5 involves a "*commit-and-proof*" framework. Here, party P_i initially commits to a random m-bit string s_i and proves in zero-knowledge about the consistency of the committed value. Each of the other parties P_j $(j \in [n] \setminus i)$ then sends a random m-bit string s_j to P_i, and the final m-bit string s is then set as the exclusive OR of all these strings. Finally P_i commits to s and proves in zero-knowledge about the consistency of both the initial and this final commitment.

However, in reality a tampered implementation of P_i might use a commitment scheme that leaks some information about s_i to an eavesdropper. The committed value might also act as a subliminal channel to leak some of its secrets (or inputs) to the other parties or to an eavesdropper. Similarly, a tampered implementation of a party P_j might also open up the possibility to leak m-bit of its input (or other secrets) to P_i or to the eavesdropper. Thus, it is desirable that the CRF resists exfiltration and also preserves security, even in the face of such a compromise. Fig. 6 shows the design of the reverse firewall for the multi-party augmented parallel coin-tossing protocol. For constructing the reverse firewall for the above protocol, we require the underlying commitment scheme and the NIZK proof system to be *malleable* (with respect to some predefined relation) and *re-randomizable*. For our application, we require that the commitment to *any* m-bit string s can be mauled to a commitment of a related but *random* m-bit string $\widehat{s} = s \oplus s'$, for any uniformly random string s'. We also require the commitment scheme to be re-randomizable, so that the randomness used to commit to a string cannot leak any information about the committed element. We show how to achieve both these properties of malleability and re-randomizability by assuming that the underlying commitment scheme com is *homomorphic* (with respect to an appropriate relation).

Our main idea is that the CRF mauls and re-randomizes the initial commitment it receives from P_i using the homomorphic properties of com. However, at this point the

Protocol: Multi-party Augmented Parallel Coin-Tossing into the Well using CRF $\mathcal{W}_1$.

Let $(\mathcal{G}, K, \mathsf{com})$ be a *perfectly hiding* and *computationally binding* commitment scheme, and $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V}, \mathsf{RandProof}, \mathsf{ZKEval})$) be a *re-randomizable* cm-NIZK *argument system* (see Def 3). Assume that P_i is the initiating party.

Party P_i | **Firewall** | **Parties $\{P_j\}_{j\in[n]\setminus i}$**

Compute (c_i, π_i),
Broadcast the tuple (c_i, π_i)

$\xrightarrow{(c_i,\pi_i)}$

Do the following:

1. Sample $s'_i \in_R \{0,1\}^m$ and $r'_i \in \mathcal{R}$
2. Compute $c'_i = \mathsf{com}_\lambda(s'_i; r'_i)$,
3. Compute $\widehat{c_i} = c_i + c'_i$,
4. Define $T_x(c_i) = \widehat{c_i} = c_i + c'_i$,
5. Compute $\widehat{\pi_i} \leftarrow \mathsf{ZKEval}'\big(\sigma_{crs}, T_x, (c_i, \pi_i)\big)$,
where $\mathsf{ZKEval}' = \mathsf{RandProof} \circ \mathsf{ZKEval}$

$\xrightarrow{(\widehat{c_i},\widehat{\pi_i})}$

$\xleftarrow{\{s_1,\cdots,s_{i-1},s_{i+1},\cdots,s_n\}}$

6. For any $j \in [n] \setminus i$, if $s_j \notin \{0,1\}^m$,
sample $s_j \xleftarrow{\$} \{0,1\}^m$.
7. Sample $\{s_i^{(j)}\}_{j\in[n]\setminus i} \xleftarrow{\$} \{0,1\}^m$,
such that $\oplus_j s_i^{(j)} = s'_i$.
8. $\forall j \in [n] \setminus i$, compute $\widehat{s_j} = s_j \oplus s_i^{(j)}$.

$\xleftarrow{\{\widehat{s_1},\cdots,\widehat{s_{i-1}},\widehat{s_{i+1}},\cdots,\widehat{s_n}\}}$

$\xrightarrow{(y,\pi)}$

9. Sample $r''_i \in \mathcal{R}$, and compute $c''_i = \mathsf{com}_\lambda(0; r''_i)$,
10. Compute $\widehat{y} = y + c''_i$,
11. Define $T'_x(c_i, (\widehat{s_1}, \cdots, \widehat{s_{i-1}}, \widehat{s_{i+1}}, \ldots, \widehat{s_n}), y) = (\widehat{c_i}, (s_1, \cdots, s_{i-1}, s_{i+1}, \ldots, s_n), \widehat{y})$
12. Compute $\widehat{\pi} \leftarrow \mathsf{ZKEval}'(\sigma_{crs}, T'_x, (c_i, (\widehat{s_1}, \cdots, \widehat{s_{i-1}}, \widehat{s_{i+1}}, \ldots, \widehat{s_n}), y), \pi)$

$\xrightarrow{(\widehat{y},\widehat{\pi})}$

Fig. 6. Reverse firewall $\mathcal{W}_1$ for the parties involved in the protocol from Fig. 5.

proof π_i given by P_i (that proves consistency of the initial commitment value) will no longer be valid with respect to the mauled commitment. Hence, the CRF also needs to *maul* the proof in such a way that the mauled proof is consistent with the mauled statement (i.,e the commitment). At first thought, it seems that the CRF cannot produce such a proof, since it does not know the witness corresponding to the original statement (i.e., the committed string and the randomness used for commitment) and hence, also has no knowledge of the mauled witness (witness resulting from mauling the statement/commitment). Fortunately, as we show, the CRF can still maul the proof π_i without actually knowing the mauled witness, thanks to the availability of the public evaluation algorithm ZKEval of the underlying controlled-malleable simulation-extractable NIZK argument system. The mauled proof is then further re-randomized using the algorithm $\mathsf{RandProof}$, so that the randomness used in the proof does not reveal any information about the witness. Finally, the resulting proof looks like a fresh proof corresponding to the mauled statement. The firewall then places the mauled commitment-proof pair on the broadcast channel. When any other party P_j sends a string s_j, the CRF checks if the string is indeed a m-bit string. If not, it chooses a random m-bit string on behalf of P_j. It then computes a $(n-1)$-out-of-$(n-1)$ secret sharing of the initial offset string s'_i chosen by the CRF at the beginning (used to maul the initial commitment) and re-randomizes all the received strings by adding the shares to them. At this point, another technical difficulty arises: the views of party P_i and all other parties in the protocol are *inconsistent* due to the above mauling by the CRF. However, as we show, the CRF can again appropriately maul the transcript (which will be treated as a statement in the final NIZK proof) so that at the end all the parties arrive at a consistent view of the protocol. The design of the reverse firewall (see Fig. 6) is now described in details:

1. The CRF $\mathcal{W}_1$ receives a commitment-proof pair (c_i, π_i) from party P_i. Let us assume that c_i is a commitment to some m-but string s_i (may not be random). It then does the following:
   - Sample another random m-bit string $s'_i \in_R \{0,1\}^m$ and a randomizer $r'_i \in_R \mathcal{R}$ for the commitment scheme com.
   - Compute $c'_i = \mathsf{com}_\lambda(s'_i, r'_i)$ and then homomorphically compute the mauled commitment $\widehat{c_i} = c_i + c'_i$.
   - Define the transformation $T_x(c_i) = \widehat{c_i} = c_i + c'_i$.
   - Derive a proof for the transformed statement as: $\widehat{\pi_i} \leftarrow \mathsf{RandProof} \circ \mathsf{ZKEval}\big(\sigma_{crs}, T_x, (c_i, \pi_i)\big)$. Note that, the proof $\widehat{\pi_i}$ is consistent with the mauled commitment $\widehat{c_i}$.
   - The firewall then places the tuple $(\widehat{c_i}, \widehat{\pi_i})$ on the broadcast channel.
2. On receiving the strings s_j from party P_j $(j \in [n] \setminus i)$, the CRF checks if $s_j \in \{0,1\}^m$. If not, then it chooses a random string $s_j \in \{0,1\}^m$. It then computes a $(n-1)$-out-of-$(n-1)$ xor secret sharing of the string s'_i, i.e., it samples random strings $s_i^{(j)}$, such that $\oplus_j s_i^{(j)} = s'_i$. Then, for all $j \in [n] \setminus i$, it computes $\widehat{s_j} = s_j \oplus s_i^{(j)}$. It then forwards the tuple $\{\widehat{s_1}, \cdots, \widehat{s_{i-1}}, \widehat{s_{i+1}}, \cdots, \widehat{s_n}\}$ to party P_i.
3. Receive the tuple (y, π) from P_i. Note that, the proof π will not be consistent with the view of the other parties $\{P_j\}_{j \in [n] \setminus i}$, since the common input (or statement) for P_j will be different from the input of party P_i. In particular, the (public) input for P_i is the tuple $(c_i, \widehat{s_1}, \cdots, \widehat{s_{i-1}}, \widehat{s_{i+1}}, \cdots, \widehat{s_n})$, while the (public) input for the parties P_j is the tuple $(\widehat{c_i}, s_1, \cdots, s_{i-1}, s_{i+1}, \cdots, s_n)$. The CRF then does the following:
   - Sample $r''_i \in_R \mathcal{R}$, and compute $c''_i = \mathsf{com}_\lambda(0; r''_i)$. Then, it homomorphically computes the re-randomized commitment $\widehat{y} = y + c''_i$.
   - Define the following transformation:

     $$T'_x(c_i, (\widehat{s_1}, \cdots, \widehat{s_{i-1}}, \widehat{s_{i+1}}, \cdots, \widehat{s_n}), y) = (\widehat{c_i}, (s_1, \cdots, s_{i-1}, s_{i+1}, \cdots, s_n), \widehat{y}).$$

     Note that, this is efficiently computable, given the knowledge of the strings $\{s_i^{(j)}\}_{j \in [n] \setminus i}$.

- Compute the proof $\widehat{\pi}$ as follows:
  $\widehat{\pi} \leftarrow \mathsf{RandProof} \circ \mathsf{ZKEval}\big(\sigma_{crs}, T'_x, (c_i, (\widehat{s_1}, \cdots, \widehat{s_{i-1}}, \widehat{s_{i+1}}, \cdots, \widehat{s_n}), y), \pi\big)$. Broadcast the tuple $(\widehat{y}, \widehat{\pi})$ to all the parties P_j. Note that, the proof $\widehat{\pi}$ is now consistent with the statement $(\widehat{c_i}, s_1, \cdots, s_{i-1}, s_{i+1}, \cdots, s_n)$.

Theorem 4. *The reverse firewall $\mathcal{W}_1$ for augmented multi-party coin-tossing shown in* Fig. 6 *is* functionality-maintaining. *If the commitment scheme* com *is* computationally binding *and is* homomorphic *with respect to the (addition) operation defined over the underlying groups (i.e, the message space, randomness space and the commitment space of* com*) and the NIZK argument system is* controlled-malleable simulation-sound extractable, *then the firewall $\mathcal{W}_1$* preserves security *for party P_j and is* weakly exfiltration-resistant *against the other parties $\{P_j\}_{j\in[n]\setminus i}$. If the commitment scheme is* perfectly/statistically hiding *and homomorphic as above and the NIZK argument system also satisfies the same property as above, $\mathcal{W}_1$* strongly preserves security *for the parties $\{P_j\}_{j\in[n]\setminus i}$ and is* strongly exfiltration-resistant *against P_i. The firewall $\mathcal{W}_1$ also* detects failures *for all the parties.*

Proof. First, we will show that the reverse firewall shown in Fig. 6 is functionality maintaining. If the parties are honest, the output view of all these parties are *consistent.* In particular, the output of party P_i is: $\widehat{s} = s_i \oplus (\widehat{s_1}, \cdots, \widehat{s_{i-1}}, \widehat{s_{i+1}}, \cdots, \widehat{s_n}) = (s_i \oplus s'_i) \oplus_{j\in[n]\setminus i} s_j$. The output of P_i is a commitment y to the m-bit string $\widehat{s}$. Even if all the strings s_i and $(s_1, \cdots, s_{i-1}, s_{i+1}, \cdots, s_n)$ are not random, the resultant m-bit string $\widehat{s}$ is indeed *random.* Hence, at the end party P_i ends up with a random pad, while the other parties receives a commitment to the string. This shows that the CRF is *functionality-maintaining.* We now proceed to show that the reverse firewall for P_i preserves security and exfiltration-resistance against the other parties $\{P_j\}_{j\in[n]\setminus i}$. Note that, the homomorphically evaluated commitment $\widehat{c_i}$ is *independent* of the original commitment c_i. This is because the firewall chooses an independent m-bit string s'_i and randomness r'_i to homomorphically evaluate the original (potentially malicious) commitment string c_i. The proof π_i is also appropriately mauled so that the mauled proof $\widehat{\pi_i}$ is consistent with the mauled commitment $\widehat{c_i}$. The mauled proof is further re-randomized using the algorithm $\mathsf{RandProof}$. Hence, by the *derivation-privacy* of the proof of the NIZK argument system (see Thm. 1), the mauled proof $\widehat{\pi_i}$ looks indistinguishable from a fresh proof of the commitment $\widehat{c_i}$. Hence, the firewall sanitizes the messages sent across by P_i, even though the implementation of P_i may be corrupt. Since, P_i is functionality maintaining, his second message is fixed, unless he can find an alternate opening for c_i, which by definition of binding is computationally hard. Hence, it follows that the reverse firewall for party P_i is weakly exfiltration-resistant for P_i against all the other parties P_j and also preserves security for P_i. To prove strong exfiltration-resistance for any party P_j against party P_i and strong security preservation for P_j, one should note that the mauled commitment is a uniformly random commitment to a uniformly random m-bit string. Since, the commitment scheme com is perfectly (statistically) hiding, it is (statistically) independent of the string s_j chosen by the party P_j. The firewall mauls all the the strings $\{s_j\}_{j\in[n]\setminus i}$ by adding a share of the random offset s'_i to each s_j (using $(n-1)$-out-of-$(n-1)$ secret sharing of s_i'), and hence the final m-bit string of party P_i is *random*, irrespective of how the strings s_j were chosen.

6.3 Multi-party Input Commitment phase using Reverse Firewalls

In this step, each party commits to its input to be used in the protocol. Similar to the augmented coin-tossing functionality (see Def. 12), we slightly *modify* the multi-party input commitment functionality to fit the framework of reverse firewalls. In particular, in the traditional multi-party input commitment functionality, the initiating party receives a tuple (x, r) and all other parties receive a commitment $\mathsf{com}(x; r)$ of the same tuple, for a statistically/perfectly hiding (and computationally binding) commitment scheme. Looking ahead, the value x would serve as the input of the initiating party in the protocol. For our purpose, we change the above functionality as follows: The initiating party receives a tuple (x, r) and all other parties receive a commitment $\mathsf{com}(x; r')$ for a potentially different r'. This means that, all the other parties still receive a commitment of the same random coin x but potentially under different randomness. More formally, the parties execute a secure protocol for the following functionality:

Definition 13 (Multi-party Input Commitment). *An n-party* input commitment *protocol is an n-party protocol for securely computing the following functionality with respect to a fixed commitment scheme $\{\mathcal{G}_\lambda, K_\lambda, \mathsf{com}_\lambda\}_{\lambda\in\mathbb{N}}$,*

$$\big((x, U_{\ell\cdot\lambda}), 1^\lambda, \cdots, 1^\lambda\big) \rightarrow \big(\lambda, \mathsf{com}_\lambda(x; U'_{\ell\cdot\lambda}), \cdots, \mathsf{com}_\lambda(x; U'_{\ell\cdot\lambda})\big). \tag{2}$$

where $x \in \{0,1\}^\ell$ is the input string of the party, U_m denotes the uniform distribution over m-bit strings, and we assume that com *requires λ random bits to commit to each bit of x. Note that the string $U'_{\ell\cdot\lambda}$ may be different from the string $U_{\ell\cdot\lambda}$.*

In the input commitment phase, each party P first chooses a random string x and commits to x using randomness r to generate the commitment C. It also generates a proof π using a simulation-extractable non-interactive zero-knowledge argument system that it knows a witness (i.,e, the tuple (x, r)) corresponding to the commitment C. Finally, party P_i places the pair (C, π) on the broadcast channel. Next, we present a reverse firewall $\mathcal{W}_2$ for the above protocol, as shown in Figure 7. As before, we assume that P_i is the initiating party.

The main idea of the working of the reverse firewall $\mathcal{W}_2$ is very simple (see Fig. 6). The CRF simply re-randomizes the commitment C_i and the proof $\varPi_i$ received from party P_i. The way the CRF re-randomizes the commitment C_i is by homomorphically adding to it a fresh commitment of the all zero string. To make the proof consistent with the mauled commitment, the firewall also mauls the NIZK proof using the public evaluation algorithm ZKEval of the underlying cm-NIZK argument system, and then re-randomizes the proof using the $\mathsf{RandProof}$ algorithm. The CRF then broadcasts the mauled commitment and the mauled proof.

Protocol: Multi-party Input-commitment using CRF $\mathcal{W}_2$.

Let $(\mathcal{G}, K, \mathsf{com})$ be a *perfectly hiding* and *computationally binding* commitment scheme (see Def. 2), and $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V}, \mathsf{RandProof}, \mathsf{ZKEval}))$ be a *re-randomizable* cm-NIZK *argument system* (see Def 3) for the following language: $\mathcal{L} = \{C, (x, y) \,|\, C = \mathsf{com}_\lambda(x; y)\}$.

Party P_i	**Firewall**	**Party $\{P_j\}_{j\in[n]\setminus i}$**
Compute $(C_i, \mathit{\Pi}_i)$, Broadcast the tuple $(C_i, \mathit{\Pi}_i)$		
$\xrightarrow{(C_i, \mathit{\Pi}_i)}$		
	Do the following: **1.** Sample $\overline{r_i} \in \mathcal{R}$ and **2.** Compute $\overline{C_i} = \mathsf{com}_\lambda(0^m; \overline{r_i})$, **3.** Compute $\widehat{C_i} = C_i + \overline{C_i}$, **4.** Define $T(C_i) = \widehat{C_i} = C_i + \overline{C_i}$, **5.** Compute $\widehat{\mathit{\Pi}_i} \leftarrow \mathsf{ZKEval}'(\sigma_{crs}, T, C_i, \mathit{\Pi}_i)$ where $\mathsf{ZKEval}' = \mathsf{RandProof} \circ \mathsf{ZKEval}$	
		$\xrightarrow{(\widehat{C_i}, \widehat{\mathit{\Pi}_i})}$

Fig. 7. Reverse Firewall $\mathcal{W}_2$ for the Multi-party Input commitment protocol

Theorem 5. *Let $\{\mathcal{G}_\lambda, K_\lambda, \mathsf{com}_\lambda\}_{\lambda\in\mathbb{N}}$ be a perfectly hiding and computationally binding commitment scheme. Also, let $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V}, \mathsf{ZKEval})$ be a simulation-extractable non-interactive zero-knowledge argument system for the language $\mathcal{L} = \{C \mid C = \mathsf{com}_\lambda(x; y)\}$. Then the protocol in* Figure 7 *securely computes the functionality presented in* Eq. 2*. The reverse firewall $\mathcal{W}_2$ shown in* Figure 7 *is* functionality-maintaining *and* detects failure *for party P_i. If the commitment scheme* com *is* perfectly hiding*,* computationally binding *and* homomorphic *with respect to the (addition) operation defined over the underlying groups (i.e, the message space, randomness space and the commitment space of* com*); the NIZK argument system is* re-randomizable *and* simulation-sound extractable*, then the reverse firewall $\mathcal{W}_2$* preserves security *for party P_i and is* exfiltration-resistant *against the other parties $\{P_j\}_{j\in[n]\setminus i}$.*

Proof. The proof that the underlying protocol is secure follows in a straightforward manner from the proof of the input commitment functionality, as presented in [17]. The only difference is that, instead of using a strong NIZK proof of knowledge (in the plain model), we use a cm-NIZK in the CRS model. However, this change is purely an efficiency concern and in no way impacts the security of the protocol. It is also clear that the firewall maintains functionality and fails detectably for party P_i. The re-randomizability property of the commitment scheme com guarantees that $\widehat{C_i}$ is indistinguishable from a fresh commitment. Similarly, the derivation privacy property of the of the cm-NIZK argument system guarantees that the mauled proof $\widehat{\mathit{\Pi}_i}$ is indistinguishable from a fresh and honest proof on the statement $\widehat{C_i}$ by the party P_i (using the appropriate mauled witness). Hence, the firewall $\mathcal{W}_2$ preserves security for P_i and resists exfiltration for P_i against the other parties $\{P_j\}_{j\in[n]\setminus i}$. □

6.4 Multi-party Authenticated Computation Protocol using Reverse Firewalls

Let $f, h : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be polynomial-time computable. The goal of this protocol is to force the initializing party P_i to compute $f(\alpha, \beta)$, where β is known to all the parties, α is known only to P_i, and $h(\alpha, r)$ (where h is one-to-one function) is known to all the parties, where r is known only to party P_i. Here f captures the desired computation. Similar to the augmented coin-tossing functionality and the multi-party input commitment functionality (see Def. 12 and Def. 13), we slightly *modify* the multi-party authenticated computation functionality to fit the framework of reverse firewalls. In particular, we require that the function h is a one-to-one *additively homomorphic* function. Further, instead of receiving $h(\alpha, r)$, all parties in this modified protocol will receive $h(\alpha, \widehat{r})$, where $\widehat{r}$ is different from r with overwhelming probability.

Looking ahead, in the actual protocol the random variable α is set to be the concatenation of the party's original input and its' random tape (obtained as output of the augmented coin-tossing protocol), r is set to be the concatenations of the randomness used in generating the commitments in the multi-party augmented coin-tossing and multi-party input commitment protocols, and $h(\alpha, r)$ denote the concatenations of the commitments themselves from the coin-tossing and input commitment protocols. However, since the final commitments in the augmented coin-tossing and input commitment protocols are re-randomized by the reverse firewall, the parties receive some other value $h(\alpha, \widehat{r})$ (instead of $h(\alpha, r)$). However, the reverse firewall can use its internal state (from the previous augmented coin-tossing and input commitment protocols) to compute $h(\alpha, \widehat{r})$ from the value $h(\alpha, r)$ using the additive homomorphic property of the function h. In particular, the parties execute this protocol for computing the following functionality:

$$\big((\alpha, r, \beta), (h(\alpha, \widehat{r}), \beta), \cdots, (h(\alpha, \widehat{r}), \beta)\big) \to \big(\lambda, f(\alpha, \beta), \cdots, f(\alpha, \beta)\big). \tag{3}$$

The Construction. The multi-party authenticated computation protocol is run by all the parties after executing the multi-party input commitment and the multi-party (augmented) coin-tossing protocols. Hence, at this point, the inputs and the random tapes of all the parties are fixed. Other than its own input and the random tape (along with other decommitment values/randomnesses), each party also holds the commitment to all the other parties input and random tapes. We now just briefly recall the multi-party authenticated computation protocol. We follow the protocol as stated in [17], except that we use cm-NIZK argument systems instead of strong zero-knowledge proof of knowledge (as in [17]). The use of NIZK arguments naturally makes the protocol constant-round, albeit with a setup assumption. Assume that the party P_i is the initiating party in a particular run of this protocol. The input to P_i is the tuple (α, r, β), while the common input to all the parties is $(\widehat{u}, \beta)$, where $\widehat{u} = h(\alpha, \widehat{r})$, and $\widehat{r}$ is different from r with overwhleming probability. Party P_i then computes the desired functionality $f(\alpha, \beta)$

Protocol: Multi-party Authenticated Computation Protocol using CRF $\mathcal{W}_3$.

Let $f, h : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be two-argument functions, and let h be a *one-one additively homomorphic* function.
Input to P_i: (α, r, β).
Common input for all the parties: $(\widehat{u}, \beta)$, where $\widehat{u} = h(\alpha, \widehat{r})$. Also, let us denote by u the value $u = h(\alpha, r)$. Note that, the firewall $\mathcal{W}_3$ possesses internal state st (st precisely consists of the randomnesses r_i'' (see Fig. 6) and $\overline{r_i}$ (see Fig. 7) sampled in the augmented coin-tossing protocol and input commitment protocol respectively) to transform u to $\widehat{u}$.
Let $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V}, \mathsf{RandProof}, \mathsf{ZKEval}))$ be a *re-randomizable* cm-NIZK *argument system* (see Def 3) for the following language: $\mathcal{L} = \{(u, v, f, h), (\alpha, r) \,|\, u = h(\alpha, r) \wedge v = f(\alpha, \beta)\}$.

Party P_i	**Firewall**	**Party $\{P_j\}_{j \in [n] \setminus i}$**
Compute (v, Π), Broadcast the tuple (v, Π)	Stores st	
$\xrightarrow{(v,\Pi)}$		
	Do the following: Compute $\widehat{u} = h(\alpha, \widehat{r})$ from u using st. Define $T(u) = \widehat{u}$. Compute $\widehat{\Pi} \leftarrow \mathsf{ZKEval}'(\sigma_{crs}, (u, T, v, f, h), \Pi)$ where $\mathsf{ZKEval}' = \mathsf{RandProof} \circ \mathsf{ZKEval}$	
		$\xrightarrow{(v,\widehat{\Pi})}$

Fig. 8. Reverse Firewall $\mathcal{W}_3$ for the Multi-party authenticated computation protocol

and invokes a cm-NIZK argument system to generate a proof Π corresponding to the following language: $\mathcal{L} = \{\big((u, v, f, h), (\alpha, r)\big) \mid \big((u = h(\alpha, r)) \wedge (v = f(\alpha, \beta))\big)\}$. It then broadcasts the tuple (v, Π). In case the proof does not verify, all the parties abort and output $\perp$.

We now discuss the design of the reverse firewall $\mathcal{W}_3$ for this protocol. We assume that the party P_i is tampered in a functionality-maintaining way. The idea for the design of the CRF is very simple: the CRF possesses state st used to transform u to the value $\widehat{u}$. It then mauls the proof Π using the cm-NIZK and also re-randomizes the mauled proof. Note that, the value $v = f(\alpha, \beta)$ given by P_i should be correctly computed. This follows from the fact that party $P_i's$ input and random coins are fixed, and it is corrupted in a functionality-maintaining way. The design of the CRF is shown in Fig. 8.

Theorem 6. *Let $\{\mathcal{G}_\lambda, K_\lambda, \mathsf{com}_\lambda\}_{\lambda \in \mathbb{N}}$ be a perfectly hiding and computationally binding commitment scheme. Also, let $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V}, \mathsf{ZKEval})$ be a strong simulation-extractable non-interactive zero-knowledge argument system for the language $\mathcal{L}$ shown in* Figure 8. *Then the protocol in* Figure 8 *securely computes the functionality presented in* Eq. 3. *The reverse firewall $\mathcal{W}_3$ shown in* Figure 8 *is functionality-maintaining and detects failure for party P_i. If the commitment scheme* com *is* perfectly hiding *and* computationally binding*; the NIZK argument system is* re-randomizable *and* simulation-sound extractable*, then the reverse firewall $\mathcal{W}_3$ preserves security for party P_i and is exfiltration-resistance against the other parties $\{P_j\}_{j \in [n] \setminus i}$.*

Proof. The proof that the underlying protocol is secure follows from the proof of the protocol executing the authenticated computation functionality, as presented in [17]. The only difference compared to [17] is that we use a cm-NIZK, instead of a strong NIZK proof of knowledge. It is also clear that the firewall maintains functionality and fails detectably for party P_i. Consider a functionality-maintaining tampered implementation of party P_i, i.e, $\widetilde{P}_i$, and let (v, Π) be the output of $\widetilde{P}_i$. Since $\widetilde{P}_i$ is functionality-maintaining, the value v should be correctly computed, and the NIZK proof Π should also comply to the fact that the computational is consistent. However, the proof Π is not consistent with the value $\widehat{u} = h(\alpha; \widehat{r})$ stoted by the other parties. Hence, the firewall mauls the proof appropriately, re-randomizes the proof and outputs the mauled proof $\widehat{\Pi}$. The derivation privacy of cm-NIZK implies that the proof $\widehat{\Pi}$ is indistinguishable from a fresh proof using the mauled witnesses. Hence, the security preservation and exfiltration-resistance follows. □

6.5 The Final Compiler

We now present the final compiler which transforms any semi-honest MPC protocol $\boldsymbol{\Pi}$ into a protocol $\boldsymbol{\Pi'}$ which is secure in the malicious model in the setting of reverse firewalls. We assume the existence of a single broadcast channel. The specification of our compiler is similar to that presented in [17]; however, adjusted to the reverse firewall setting. In particular, we present a reverse firewall $\mathcal{W}^*$ for the final MPC protocol $\boldsymbol{\Pi'}$. As we show, this firewall $\mathcal{W}^*$ can be seen as consisting of three sub-firewalls $\mathcal{W}_1$, $\mathcal{W}_2$ and $\mathcal{W}_3$ corresponding to the three sub-protocols or building blocks used in the compiler, namely, input commitment, (augmented) coin-tossing, and the authenticated computation protocols respectively. We then present a generic composition theorem for reverse firewalls and show that the compiled protocol $\boldsymbol{\Pi'}$ is secure in the presence of the reverse firewall $\mathcal{W}^*$.

The Construction. Let $\boldsymbol{\Pi}$ be a given n-party MPC protocol, secure in the semi-honest model. We compile the protocol $\boldsymbol{\Pi}$ into another protocol $\boldsymbol{\Pi'}$ in the reverse firewall setting using the building blocks we have developed so far. The specification of the protocol $\boldsymbol{\Pi'}$ follows:

Inputs. Party P_i gets input $x^i = x^i_1 x^i_2 \cdots x^i_\ell \in \{0,1\}^\ell$.

Input Commitment phase using reverse firewalls. Each of the n parties commits to their ℓ-bit input string using a secure implementation of the multi-party input commitment functionality (see Eq. 2) using reverse firewall $\mathcal{W}_1$, as presented in Fig. 7. That is, for all $j \in [n]$, party P_j selects $r^j \in \{0,1\}^\ell$ and invokes a secure implementation of the multi-party input commitment protocol using reverse firewall $\mathcal{W}_1$, playing the role of the (initializing) party P_i with input (x^j, r^j). The other parties play the role of other parties $\{P_k\}_{k \in [n] \setminus i}$ of Fig. 7 with input 1^λ, and obtain the output $\mathsf{com}_\lambda(x^j; r^j)$. Party i records r^j, and the other parties record $\mathsf{com}_\lambda(x^j; r^j)$.

Coin-generation phase. Each of the n parties run a secure implementation of the multi-party augmented parallel coin-tossing functionality (see Eq. 1) using reverse firewall $\mathcal{W}_2$, as presented in Fig. 6. This protocol is run by each party to generate a random pad of length t for emulation of the corresponding party in the semi-honest MPC protocol $\boldsymbol{\Pi}$. The other parties obtain a commitment of the random tape of that party. That is, for all $j \in [n]$, party P_j invokes a secure implementation of the multi-party augmented parallel coin-tossing protocol using reverse firewall $\mathcal{W}_2$ (see Fig. 6), playing the role of party P_i with input 1^λ. The other parties play the role of parties $\{P_k\}_{k\in[n]\setminus j}$ of Fig. 6. Party P_j obtains a pair (s^j, ω^j), where $s^j \in \{0,1\}^t$ and $\omega^j \in \{0,1\}^{t\cdot\lambda}$. The other parties obtain the commitment $\mathsf{com}_\lambda(s^j; \omega^{j'})$, where $\omega^{j'}$ may be different from the string ω^j. Party P_j records (s^j, ω^j), and the other parties record $\mathsf{com}_\lambda(s^j; \omega^{j'})$.

Protocol emulation phase. Each of the n parties run a secure implementation of the multi-party authenticated computation functionality (see Eq. 3) using reverse firewall $\mathcal{W}_3$ as presented in Fig. 8. The party which is supposed to send a message plays the role of party P_i in Eq. 3 and all the other parties play the role of other parties $\{P_k\}_{k\in[n]\setminus i}$. The variables α, β, r, and the functions h, f of the protocol are set as follows. The string α is set to be the concatenations of the party's original input and its' random tape. The string r is set to be the concatenations of all the randomnesses used to generate the commitments, and $h(\alpha, \widehat{r})$ is set to be the concatenations of the commitments themselves as received by the other parties (note that $\widehat{r}$ will be different from r with overwhelming probability).

$$\begin{aligned} \alpha &= (x^i, s^i), \;\; \text{where } x^i = x^i_1 x^i_2 \cdots x^i_\ell, \;\; \text{and } s^i \in \{0,1\}^t, \\ r &= \big(r^i, \omega^i\big), \;\; \text{where } r^i \in \{0,1\}^{\ell\cdot\lambda}, \omega_i \in \{0,1\}^{t\cdot\lambda}, \\ h(\alpha, \widehat{r}) &= \big(\mathsf{com}_\lambda(x^i; r^{i''}), \mathsf{com}_\lambda(s^i; \overline{\omega^i})\big) \end{aligned}$$

The string β is set to be the concatenation of all previous messages sent by other parties over the broadcast channel. Finally, the function f is set to be the *next message function*, i.e, the computation that determines the next message to be sent by P_i in $\boldsymbol{\Pi}$. The message can be thought of as a deterministic polynomial-time computable function of the party's input, it's random pad and the messages received so far.

Aborting. We denote the composed firewall for the compiled protocol as $\mathcal{W}^*$. The reverse firewall $\mathcal{W}^*$ is composed of three sub-firewalls $\mathcal{W}_1$, $\mathcal{W}_2$ and $\mathcal{W}_3$ corresponding to the three sub-protocols or building blocks as mentioned above. In case, any of these sub-firewalls *fails detectably*, the firewall $\mathcal{W}^*$ for the larger protocol also aborts the execution and outputs $\perp$. Else, the outputs are as follows:

Output. At the end of the protocol emulation phase, each party holds locally its output value. The parties simply output their respective values.

The composition theorem below shows that the final compiled protocol $\boldsymbol{\Pi'}$ is an actively-secure MPC protocol. The protocol $\boldsymbol{\Pi'}$ has a reverse firewall for all parties provided that each of the input commitment, the (augmented) coin-tossing and the authenticated computation protocols have their own firewalls satisfying some properties.

Theorem 7. (Composition Theorem for security of $\boldsymbol{\Pi'}$). *Given a MPC protocol $\boldsymbol{\Pi}$ secure in the semi-honest model, and provided that the multi-party input commitment* *protocol $\boldsymbol{\Pi}'_1$, the multi-party (augmented) coin-tossing protocol $\boldsymbol{\Pi}'_2$, and the multi-party authenticated computation protocol $\boldsymbol{\Pi}'_3$ are secure in the malicious model, the compiled MPC protocol $\boldsymbol{\Pi}'$ is an actively-secure MPC protocol. Let $\mathcal{W}^*_1$, $\mathcal{W}^*_2$ and $\mathcal{W}^*_3$ denote the reverse firewalls for the protocols $\boldsymbol{\Pi}'_1$, $\boldsymbol{\Pi}'_2$ and $\boldsymbol{\Pi}'_3$ respectively. Also, let party P_i be the initiating party for all these protocols at some point in time (in general it can be any one of the parties corrupted in a functionality-maintaining way). Now consider the following properties:*

- *Let $\boldsymbol{\Pi}$ be a MPC protocol secure in the semi-honest model (without reverse firewalls).*
- *Let the firewall $\mathcal{W}^*_1$ (for the multi-party input commitment protocol $\boldsymbol{\Pi}'_1$)* preserves security *for party P_i, is* exfiltration-resistant *against the other parties $\{P_j\}_{j\in[n]\setminus i}$, and* detects failure *for P_i.*
- *Let the firewall $\mathcal{W}^*_2$ (for the multi-party augmented coin-tossing protocol $\boldsymbol{\Pi}'_2$)* preserves security *for party P_i and is* weakly exfiltration-resistant *against the other parties $\{P_j\}_{j\in[n]\setminus i}$. Also, let $\mathcal{W}_2$ strongly preserve the security for the parties $\{P_j\}_{j\in[n]\setminus i}$ and is strongly exfiltration-resistant against P_i. Finally, let $\mathcal{W}_2$* detect failures *for all the parties.*
- *Let the firewall $\mathcal{W}^*_3$ (for multi-party authenticated computation protocol $\boldsymbol{\Pi}'_3$)* preserves security *for party P_i, is* weakly exfiltration-resistant *against the other parties $\{P_j\}_{j\in[n]\setminus i}$, and detects failure for P_i.*

Then the composed reverse firewall $\mathcal{W}^* = \mathcal{W}^*_1 \circ \mathcal{W}^*_2 \circ \mathcal{W}^*_3$ preserves security *for party P_i and is* weakly exfiltration-resistant *against the parties $\{P_j\}_{j\in[n]\setminus i}$ in the protocol $\boldsymbol{\Pi}'$.*

Proof. The proof for the first part of the composition theorem is a very well-known result. We refer the reader to [17] for the detailed proof. Before proceeding with the proof of the second part, we introduce some additional notations. Let $\widetilde{P}_i$ denote the functionality-maintaining tampered implementation of party P_i in the final protocol $\boldsymbol{\Pi}'$. For two distinct protocols $\boldsymbol{\Pi}_j$ and $\boldsymbol{\Pi}_k$, we denote by $\boldsymbol{\Pi}_j \circ \boldsymbol{\Pi}_k$ the composed protocol obtained by running the two protocols sequentially, i.e, running the protocol $\boldsymbol{\Pi}_j$ followed by the protocol $\boldsymbol{\Pi}_k$. Let $\widetilde{P'_i}$ be the truncation of $\widetilde{P}_i$ till the end of the execution of the protocol $\boldsymbol{\Pi}'_1 \circ \boldsymbol{\Pi}'_2$ (i.,e, till the execution of the (augmented) coin-tossing protocol). Also, let $\widetilde{P''_i}$ be the truncation of $\widetilde{P}_i$ to the end of the execution of the protocol $\boldsymbol{\Pi}'_1$ (i.e,. till the execution of the multi-party input-commitment protocol). We follow the game hopping technique to prove the theorem. Consider the following sequence of games.

Game 1 corresponds to the security model for MPC protocols in the malicious model, except that we replace party P_i with the composed party $\mathcal{W}^* \circ \widetilde{P}_i$ (where $\mathcal{W}^*$ is the firewall for the final protocol $\boldsymbol{\Pi}'$). **Game 2** is similar to Game 1, except that we consider the protocol $\boldsymbol{\Pi}'$ where the multi-party input commitment protocol $\boldsymbol{\Pi}'_1$ is replaced by its corresponding ideal functionality. **Game 3** is similar to Game 2, except that we replace the composed party $\mathcal{W}^*_1 \circ \widetilde{P''_i}$ with $\mathcal{W}^*_1 \circ P''_i$ (i.e, the honest implementation of the "truncated" party P''_i (till the end of the multi-party input commitment phase) composed with the firewall $\mathcal{W}_1^*$). **Game 4** is similar to Game 3, except that we consider the protocol $\boldsymbol{\Pi}'$ where the multi-party augmented parallel coin-tossing protocol $\boldsymbol{\Pi}'_2$ is replaced by its corresponding ideal functionality. **Game 5** is similar to Game 4, except that we replace the composed party $\mathcal{W}_2^* \circ \widetilde{P'_i}$ with $\mathcal{W}_2^* \circ P'_i$ (i.e, the honest implementation of the "truncated" party P'_i (till the end of the multi-party augmented coin-tossing phase) composed with the firewall $\mathcal{W}_2^*$). **Game 6** is similar to Game 5, except that we consider the protocol $\boldsymbol{\Pi}'$ where the multi-party authenticated computation protocol $\boldsymbol{\Pi}'_3$ is replaced by its corresponding ideal functionality. **Game 7** is similar to Game 6, except that we replace the party $\mathcal{W}^* \circ \widetilde{P_i}$ with the party $\mathcal{W}^* \circ P_1$ (i.e, the honest implementation of P_i composed with the firewall $\mathcal{W}^*$).

Note that, at the end of Game 7, the advantage of any PPT adversary is negligible. This is because, when we replace the tampered implementation of the party P_i with its corresponding honest implementation, we can invoke the security of the underlying actively-secure MPC protocol mimicking $\mathcal{W}^* \circ P_i$ as the honest party (where the firewall $\mathcal{W}^*$ does nothing and simply lets all the messages pass from P_i to the other parties). Then the real world-ideal world security of MPC protocols in the semi-honest model guarantees that the advantage of any PPT adversary is negligible. Also, if there are more than one honest parties, the security of the resulting protocol naturally generalizes, since we have shown it to hold for an arbitrary party corrupted in a functionality-maintaining way. To complete the proof we show a sequence of claims that prove that every Game i is indistinguishable from Game $i+1$.

Claim 1 *If the firewall $\mathcal{W}_1^*$ for the protocol $\boldsymbol{\Pi}'_1$ preserves security for party P_i and fails detectably, then for any PPT adversary $\mathcal{A}_1$, it holds that:*

$$|\mathsf{Adv}_{\mathcal{A}_1}^{Game_1}(\lambda) - \mathsf{Adv}_{\mathcal{A}_1}^{Game_2}(\lambda)| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose the claim does not hold. Then we can construct another adversary $\mathcal{A}'_1$ against the multi-party input commitment protocol using $\mathcal{A}_1$ as a black-box. We can view the final protocol $\boldsymbol{\Pi}'$ as the composition of the sub-protocols $\boldsymbol{\Pi}'_1$, $\boldsymbol{\Pi}'_2$ and $\boldsymbol{\Pi}'_3$. Hence, we can view the party $\mathcal{W}^* \circ \widetilde{P_i}$ in the final protocol $\boldsymbol{\Pi}'$ as the composition of $\big((\mathcal{W}_1^* \circ \widetilde{P''_i}) \circ (\mathcal{W}_2^* \circ \widetilde{P'_i}) \circ (\mathcal{W}_3^* \circ \widetilde{P_i})\big)$. After running the input commitment protocol $\boldsymbol{\Pi}'_1$, the firewall $\mathcal{W}_1^*$ checks if the transcript of the underlying protocol is valid for $\mathcal{W}_1^* \circ P''_i$. Note that this can be done by running the efficient algorithm that can distinguish between valid and invalid runs of the protocol. Now, since the firewall preserves security for party P_i in the protocol $\boldsymbol{\Pi}'_1$, we can replace the protocol $\boldsymbol{\Pi}'_1$ with its corresponding ideal functionality in the malicious model, call it $\mathcal{F}_{\mathsf{comm}}$. Hence, if any adversary can distinguish between Game 1 and Game 2 with noticeable advantage, it can be used as a black-box to build a distinguisher $\mathcal{A}'_1$ between the protocol $\boldsymbol{\Pi}'_1$ and its corresponding ideal-world functionality $\mathcal{F}_{\mathsf{comm}}$, which contradicts the security of $\boldsymbol{\Pi}'_1$ in the malicious model. The claim follows. □

Claim 2 *If the firewall $\mathcal{W}_1^*$ for the protocol $\boldsymbol{\Pi}'_1$ is exfiltration-resistant for party P_i against other parties $\{P_j\}_{j\in[n]\setminus i}$, then for any PPT adversary $\mathcal{A}'_1$, it holds that:*

$$|\mathsf{Adv}^{Game_2}_{\mathcal{A}'_1}(\lambda) - \mathsf{Adv}^{Game_3}_{\mathcal{A}'_1}(\lambda)| \leq \mathsf{negl}(\lambda)$$

Proof. The protocol $\boldsymbol{\Pi}'_1$ is replaced with its ideal functionality $\mathcal{F}_{\mathsf{comm}}$ (in the presence of reverse firewalls). By definition, there is no extra leakage in the ideal world functionality beyond what is leakage by the protocol output. Hence, the tampered implementation $\widetilde{P''_i}$ (recall that P''_i is the truncation of party P_i till the end of the input commitment phase) does not leak any information through the firewall $\mathcal{W}^*_1$, due to the exfiltration-resistant property of $\mathcal{W}^*_1$. The claim follows. □

Claim 3 *If the firewall $\mathcal{W}^*_2$ for the protocol $\boldsymbol{\Pi}'_2$ preserves security for party P_i and fails detectably, then for any PPT adversary $\mathcal{A}_2$, it holds that:*

$$|\mathsf{Adv}^{Game_3}_{\mathcal{A}_2}(\lambda) - \mathsf{Adv}^{Game_4}_{\mathcal{A}_2}(\lambda)| \leq \mathsf{negl}(\lambda).$$

Proof. As before, we can view the composed party $\mathcal{W}^* \circ \widetilde{P}_i$ in the protocol $\boldsymbol{\Pi}'$ as $\big((\mathcal{W}^*_1 \circ \widetilde{P''_i}) \circ (\mathcal{W}^*_2 \circ \widetilde{P'_i}) \circ (\mathcal{W}^*_3 \circ \widetilde{P}_i)\big)$. However, before the beginning of this game, the multi-party input commitment protocol has already been replaced by its corresponding ideal functionality $\mathcal{F}_{\mathsf{comm}}$. Hence, the composed party $\mathcal{W}^* \circ \widetilde{P}_i$ can be seen as the composition of $\big(\mathcal{F}_{\mathsf{comm}} \circ (\mathcal{W}^*_2 \circ \widetilde{P'_i}) \circ (\mathcal{W}^*_3 \circ \widetilde{P}_i)\big)$. In this game, we replace the protocol $\boldsymbol{\Pi}'_2$ with its corresponding ideal functionality, call it $\mathcal{F}_{\mathsf{coin}}$. Any adversary $\mathcal{A}_2$ that can distinguish between these two games Game 2 and Game 3 can be used as a black-box to construct another distinguisher for the real world-ideal world security of the protocol $\boldsymbol{\Pi}'_2$. This follows from the fact that the firewall $\mathcal{W}^*_2$ preserves security for party P_i in the protocol $\boldsymbol{\Pi}'_2$ and also fails detectably. □

Claim 4 *If the firewall $\mathcal{W}^*_2$ for the protocol $\boldsymbol{\Pi}'_2$ is exfiltration-resistant for party P_i against other parties $\{P_j\}_{j\in[n]\setminus i}$, then for any PPT adversary $\mathcal{A}_2$, it holds that:*

$$|\mathsf{Adv}^{Game_4}_{\mathcal{A}_2}(\lambda) - \mathsf{Adv}^{Game_5}_{\mathcal{A}_2}(\lambda)| \leq \mathsf{negl}(\lambda).$$

Proof. The proof of this claim follows from the fact that the protocol $\boldsymbol{\Pi}'_2$ is replaced by its corresponding ideal functionality $\mathcal{F}_{\mathsf{coin}}$ and no information other than what is implied by the outputs of the protocol are leaked by the ideal functionality. Hence, the tampered implementation $\widetilde{P'_i}$ (recall that P'_i is the truncation of party P_i till the end of the coin-tossing phase) does not leak any information through the firewall $\mathcal{W}^*_2$, due to the exfiltration-resistant property of $\mathcal{W}^*_2$. The claim follows. □

Claim 5 *If the firewall $\mathcal{W}^*_3$ for the protocol $\boldsymbol{\Pi}'_3$ preserves security for party P_i and fails detectably, then for any PPT adversary $\mathcal{A}_3$, it holds that:*

$$|\mathsf{Adv}^{Game_5}_{\mathcal{A}_3}(\lambda) - \mathsf{Adv}^{Game_6}_{\mathcal{A}_3}(\lambda)| \leq \mathsf{negl}(\lambda).$$

Proof. At the beginning of this game, the multi-party input commitment protocol $\boldsymbol{\Pi}'_1$ and the multi-party augmented parallel coin-tossing protocol $\boldsymbol{\Pi}'_2$ has been replaced by the corresponding ideal functionalities $\mathcal{F}_{\mathsf{comm}}$ and $\mathcal{F}_{\mathsf{coin}}$ respectively in the malicious model. Hence, at this point, the inputs and the random tapes of all the parties are fixed. The composed party $\mathcal{W}^* \circ \widetilde{P}_i$ can be seen as the composition of $\big(\mathcal{F}_{\mathsf{comm}} \circ \mathcal{F}_{\mathsf{coin}} \circ (\mathcal{W}_3^* \circ \widetilde{P}_i)\big)$. In this game, we replace the protocol $\boldsymbol{\Pi}'_3$ with its corresponding ideal functionality, call it $\mathcal{F}_{\mathsf{auth\text{-}com}}$. Any adversary $\mathcal{A}_3$ that can distinguish between Game 5 and Game 6 can be used as a black-box to construct another distinguisher for the real world-ideal world security of the protocol $\boldsymbol{\Pi}'_3$. This follows from the fact that the firewall $\mathcal{W}_3^*$ preserves security for party P_i in the protocol $\boldsymbol{\Pi}'_3$ and also fails detectably. □

Claim 6 *For any PPT adversary $\mathcal{A}^*$, it holds that:*

$$|\mathsf{Adv}_{\mathcal{A}^*}^{Game_6}(\lambda) - \mathsf{Adv}_{\mathcal{A}^*}^{Game_7}(\lambda)| \leq \mathsf{negl}(\lambda).$$

Proof. Before the beginning of this game, all the sub-protocols $\boldsymbol{\Pi}'_1$, $\boldsymbol{\Pi}'_2$ and $\boldsymbol{\Pi}'_3$ are replaced with their respective ideal functionalities $\mathcal{F}_{\mathsf{comm}}$, $\mathcal{F}_{\mathsf{coin}}$ and $\mathcal{F}_{\mathsf{auth\text{-}com}}$ respectively. Hence the security of the final protocol $\boldsymbol{\Pi}'$ can be reduced to the security of the semi-honest protocol $\boldsymbol{\Pi}$. Hence, the protocol $\boldsymbol{\Pi}'$ cannot leak any additional information about the secrets of party P_i. Hence, the exfiltration-resistance of the composed firewall $\mathcal{W}^*$ follows, and we can replace $\mathcal{W}^* \circ \widetilde{P}_i$ with the party $\mathcal{W}^* \circ P_i$. □

Combining claims 1 - 6, we get the proof of Thm. 7. □

7 Conclusion and Future Work

In this work, we present the first feasibility result for general MPC protocols in the setting of reverse firewalls. Previous work in this area constructed reverse firewalls either for arbitrary 2-party functionalities in the passive setting, or for a few concrete functionalities ranging from signature schemes, oblivious transfer and key exchange protocols. Our result is obtained by revisiting the classical compiler of Goldreich, Micali, and Widgerson and making it reverse-firewall compatible. We leave open the construction of more efficient and round-optimal RF-compatible MPC protocols for future work. Constructing RF-compatible adaptively secure MPC protocols is also an interesting open problem. As mentioned in the introduction, another research direction is to develop concrete instantiations of firewalls for threshold cryptography schemes.

References

[1] P. Ananth, A. R. Choudhuri, and A. Jain. "A New Approach to Round-Optimal Secure Multiparty Computation". In: Santa Barbara, CA, USA, 2017.

[2] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. "Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE". In: *Advances in Cryptology – EUROCRYPT 2012*. Cambridge, UK, 2012.

[3] G. Ateniese, B. Magri, and D. Venturi. “Subversion-Resilient Signature Schemes”. In: *ACM CCS 15: 22nd Conference on Computer and Communications Security*. Denver, CO, USA, 2015.

[4] D. Beaver. “Foundations of Secure Interactive Computing”. In: *Advances in Cryptology – CRYPTO’91*. Santa Barbara, CA, USA, 1992.

[5] F. Benhamouda and H. Lin. “k-Round Multiparty Computation from k-Round Oblivious Transfer via Garbled Interactive Circuits”. In: 2018.

[6] M. Blum, P. Feldman, and S. Micali. “Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)”. In: *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA, 1988.

[7] R. Canetti. “Security and Composition of Multiparty Cryptographic Protocols”. In: *Journal of Cryptology* 1 (2000).

[8] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. “Malleable Proof Systems and Applications”. In: *Advances in Cryptology – EUROCRYPT 2012*. Cambridge, UK, 2012.

[9] R. Chen, Y. Mu, G. Yang, W. Susilo, F. Guo, and M. Zhang. “Cryptographic Reverse Firewall via Malleable Smooth Projective Hash Functions”. In: *Advances in Cryptology – ASIACRYPT 2016*. Berlin, Heidelberg, 2016.

[10] R. Cramer, I. B. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. 2015.

[11] E. Dauterman, H. Corrigan-Gibbs, D. Mazières, D. Boneh, and D. Rizzo. “True2F: Backdoor-Resistant Authentication Tokens”. In: *2019 IEEE Symposium on Security and Privacy*. 2019.

[12] Y. Dodis, I. Mironov, and N. Stephens-Davidowitz. “Message Transmission with Reverse Firewalls—Secure Communication on Corrupted Machines”. In: *Advances in Cryptology – CRYPTO 2016, Part I*. Santa Barbara, CA, USA, 2016.

[13] J. Doerner, Y. Kondi, E. Lee, and a. shelat. “Threshold ECDSA from ECDSA Assumptions: The Multiparty Case”. In: *2019 IEEE Symposium on Security and Privacy*. 2019.

[14] D. Dolev and H. Strong. “Authenticated Algorithms for Byzantine Agreement”. In: *SIAM Journal on Computing* 4 (1983). eprint: https://doi.org/10.1137/0212045.

[15] S. Garg and A. Srinivasan. “Two-Round Multiparty Secure Computation from Minimal Assumptions”. In: 2018.

[16] R. Gennaro and S. Goldfeder. “Fast Multiparty Threshold ECDSA with Fast Trustless Setup”. In: *ACM CCS 18: 25th Conference on Computer and Communications Security*. 2018.

[17] O. Goldreich. *Secure Multi-Party Computation*. manuscript available at http://www.wisdom.weizmann.ac.il/~oded/pp.html.

[18] O. Goldreich, S. Micali, and A. Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *19th Annual ACM Symposium on Theory of Computing*. New York City, NY, USA, 1987.

[19] J. Groth. *Homomorphic Trapdoor Commitments to Group Elements*. Cryptology ePrint Archive, Report 2009/007. `http://eprint.iacr.org/2009/007`. 2009.

[20] J. Groth. "Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures". In: *Advances in Cryptology – ASIACRYPT 2006*. Shanghai, China, 2006.

[21] D. Hofheinz, T. Jager, and E. Knapp. "Waters Signatures with Optimal Security Reduction". In: *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*. Darmstadt, Germany, 2012.

[22] Y. Ishai, A. Sahai, and D. Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *Advances in Cryptology – CRYPTO 2003*. Santa Barbara, CA, USA, 2003.

[23] Y. Lindell. "Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation". In: *Advances in Cryptology – CRYPTO 2001*. Santa Barbara, CA, USA, 2001.

[24] S. Micali and L. Reyzin. "Physically Observable Cryptography (Extended Abstract)". In: *TCC 2004: 1st Theory of Cryptography Conference*. Cambridge, MA, USA, 2004.

[25] I. Mironov and N. Stephens-Davidowitz. "Cryptographic Reverse Firewalls". In: *Advances in Cryptology – EUROCRYPT 2015, Part II*. Sofia, Bulgaria, 2015.

[26] P. Mukherjee and D. Wichs. "Two Round Multiparty Computation via Multi-key FHE". In: *Advances in Cryptology – EUROCRYPT 2016, Part II*. Vienna, Austria, 2016.

[27] G. J. Simmons. "The Prisoners' Problem and the Subliminal Channel". In: *Advances in Cryptology – CRYPTO'83*. Santa Barbara, CA, USA, 1983.

[28] B. R. Waters. "Efficient Identity-Based Encryption Without Random Oracles". In: *Advances in Cryptology – EUROCRYPT 2005*. Aarhus, Denmark, 2005.