

Reverse Firewalls for Adaptively Secure MPC without Setup

Suvradip Chakraborty¹, Chaya Ganesh², Mahak Pancholi³, and Pratik Sarkar⁴

¹ Institute of Science and Technology Austria

suvradip.chakraborty@ist.ac.at

² Indian Institute of Science, India

chaya@iisc.ac.in

³ Department of Computer Science, Aarhus University, Aarhus, Denmark.**

mahakp@cs.au.dk

⁴ Department of Computer Science, Boston University

pratik93@bu.edu

Abstract. We study Multi-party computation (MPC) in the setting of subversion, where the adversary tampers with the machines of honest parties. Our goal is to construct actively secure MPC protocols where parties are corrupted adaptively by an adversary (as in the standard adaptive security setting), and in addition, honest parties' machines are compromised.

The idea of reverse firewalls (RF) was introduced at EUROCRYPT'15 by Mironov and Stephens-Davidowitz as an approach to protecting protocols against corruption of honest parties' devices. Intuitively, an RF for a party \mathcal{P} is an external entity that sits between \mathcal{P} and the outside world and whose scope is to sanitize \mathcal{P} 's incoming and outgoing messages in the face of subversion of their computer. Mironov and Stephens-Davidowitz constructed a protocol for passively-secure two-party computation. At CRYPTO'20, Chakraborty, Dziembowski and Nielsen constructed a protocol for secure computation with firewalls that improved on this result, both by extending it to *multi*-party computation protocol, and considering *active* security in the presence of *static* corruptions.

In this paper, we initiate the study of RF for MPC in the *adaptive* setting. We put forward a definition for adaptively secure MPC in the reverse firewall setting, explore relationships among the security notions, and then construct reverse firewalls for MPC in this stronger setting of adaptive security. We also resolve the open question of Chakraborty, Dziembowski and Nielsen by removing the need for a trusted setup in constructing RF for MPC.

Towards this end, we construct reverse firewalls for adaptively secure augmented coin tossing and adaptively secure zero-knowledge protocols and obtain a constant round adaptively secure MPC protocol in the reverse firewall setting without setup. Along the way, we propose a new multi-party adaptively secure coin tossing protocol in the plain model, that is of independent interest.

** Work partially done while at Indian Institute of Science, India.

Table of Contents

Reverse Firewalls for Adaptively Secure MPC without Setup	1
<i>Suvradip Chakraborty, Chaya Ganesh, Mahak Pancholi, and Pratik Sarkar</i>	
1 Introduction	1
1.1 Our Results	2
1.2 Technical Overview	4
Adding reverse firewalls.	4
Adaptively secure coin tossing in the plain model.	6
1.3 Other Related Work	7
2 Preliminaries	8
2.1 Bilinear Groups and Knowledge of Exponent Assumption [AF07].	8
2.2 Public Key Encryption Schemes	8
2.3 Commitment Schemes	9
2.4 Zero Knowledge	11
2.5 Coin-Tossing	11
2.6 Adaptively Secure Multi-Party Computation	11
3 Reverse Firewalls for Statically secure MPC	13
4 Reverse Firewalls for Adaptively secure MPCs	14
5 Relations between Security Preservation and Exfiltration Resistance (Static case)	17
5.1 Exfiltration-Resistance implies Security Preservation	18
6 Relations between Security Preservation and Exfiltration Resistance (adaptive case) ..	20
6.1 Exfiltration-Resistance implies Security Preservation	20
7 Adaptively-Secure Compiler using Reverse Firewalls in the urs model.	24
7.1 Adaptively-Secure Augmented Coin-Tossing using Reverse Firewalls in the urs model	24
7.2 Adaptively-Secure ZK in the urs_{zk} model	27
ZK Protocol of [CSW20b]	28
Our construction	29
7.3 Adaptively-Secure MPC in the urs_{mpc} model	33
8 Adaptively-Secure Multi-party Coin-Tossing Protocol with Reverse Firewalls in the Plain Model	40
9 The Final Compiler	48

1 Introduction

The standard definitions of security in cryptographic protocols are under the assumption that honest parties can completely trust the machines that implement their algorithms. However, such an assumption may be unwarranted in the real world. The security guarantees of cryptosystems depend on the adversarial model which, however, often makes idealized assumptions that are not always realized in actual implementations. Several practical attacks in the real world exploit *implementation* details of an algorithm rather than treating it as a “black-box”. In addition, users may be forced to use hardware built by companies with expertise, and software that are mandated by standardization agencies. The capability of the adversary to “tamper” with the implementation is not captured by security models in classical cryptography. This model is not overkill, as we now know by Snowden revelations [BBG⁺13] that one of potential mechanisms for large scale mass surveillance is compromise of security by subversion of cryptographic standards, and tampering of hardware. The threat of an adversary modifying the implementation so that the subverted algorithm remains indistinguishable from the specification in black-box behavior, while leaking secrets was originally studied by Young and Yung as kleptography [YY96], and in the setting of subliminal channels by Simmons [Sim84]. Since Snowden revelations brought to light actual deployment of such attacks, there is renewed attention, and has led cryptographers to model such tampering in the security definition in order to closely capture real-world concerns.

Reverse Firewalls. The *cryptographic reverse firewall* (RF) framework was introduced by Mironov and Stephens-Davidowitz [MS15] in the context of designing protocols secure against adversaries that can corrupt users’ machines in order to compromise their security. A reverse firewall for a party \mathcal{P} is an external intermediate machine that modifies the incoming and outgoing messages sent by \mathcal{P} ’s machine. In essence, a reverse firewall sits between a party \mathcal{P} and the external world, and “sanitizes” the messages that are sent and received by \mathcal{P} . Note that the party does not put any trust in the RF, meaning that it does not share any secrets with the firewall. This rules out trivial solutions like a trusted RF that simply keeps \mathcal{P} ’s secrets and runs on \mathcal{P} ’s behalf. Instead, the goal is for an uncorrupted⁵ RF to provide meaningful security guarantees even in the case that an honest party’s machine has been tampered with. Consider an arbitrary protocol that satisfies some notions of functionality and security. A reverse firewall for a protocol is said to *functionality-maintaining* if the resulting protocol (protocol with a firewall for party \mathcal{P}) achieves the same functionality as the original protocol. Roughly, the RF should not ruin the functionality of the underlying protocol, in the sense that the protocol with an RF for a party should still work as expected in case no subversion takes place. At the same time, the RF is expected to preserve security. An RF is said to preserve security if the protocol *with* the firewall is secure even when an honest party’s implementation is tampered with to behave in an arbitrarily corrupt way. Finally, an RF should provide *exfiltration-resistance*, i.e., regardless of how the user’s machine behaves, the presence of the RF will prevent the machine from leaking any information to the outside world.

The work of [MS15] provides a construction of a two-party passively secure computation protocol with a reverse firewall. The recent work of [CDN20] generalizes reverse firewalls for secure computation by showing feasibility of reverse firewalls for Multi Party Computation (MPC). They give a construction of reverse firewalls for secure computation in a stronger and general setting that handles multiple parties, and consider protocols in the malicious security model.

⁵ The RF being corrupt is not interesting in the active setting, since the corrupt RF and the other party together can be thought of as the adversary.

RFs in other settings have been constructed including key exchange and secure message transmission [DMS16, CMY⁺16], oblivious transfer [DMS16, CMY⁺16], digital signatures [AMV15], and zero-knowledge proofs (ZK) [GMV20]. Reverse firewalls has also been used in a practical context in the design of *True2F* [DCM⁺19], a system that is based on a firewalled key generation and ECDSA signature generation with potential applications in cryptocurrency wallets.

Besides the reverse firewall framework, other directions have been explored to address the challenge of protecting cryptosystems against different forms of subversion. We review some of them in Sec. 1.3.

1.1 Our Results

In this work, we take forward the study of reverse firewalls in the setting of MPC. We begin by proposing definitions that capture the requirements of an RF for MPC in the presence of *adaptive* corruptions. We then explore relationships among them the notions. Next, we turn our attention to constructing RFs for maliciously secure protocols in the presence of adaptive corruptions. Towards this end, we construct protocols with reverse firewalls for multi-party augmented coin tossing, zero-knowledge, and coin tossing, all in the presence of adaptive corruptions. We then use the above building blocks to construct a maliciously secure MPC in the presence of adaptive corruptions together with a reverse firewall. We further elaborate on the contributions in this work.

On the relationship between definitions. As our first contribution, we revisit the different notions of subversion security for MPC protocols in the presence of RF. The work of [MS15] defined the notions of *security preservation* (SP) and *exfiltration resistance* (ER) as the properties required from an RF. SP asks that an RF preserve the security properties of the underlying protocol for an honest party even when the honest party’s implementation is tampered with. ER is concerned with a type of attack called exfiltration, where an honest party’s tampered implementation attempts to leak secrets. A reverse firewall that is exfiltration resistant prevents an adversary from learning secrets even when the honest party’s machine is tampered with. Roughly, exfiltration resistance for a party P_i asks that the transcripts produced in the following two ways are *indistinguishable*: (i) by running the protocol with the RF for P_i whose implementation has been arbitrarily subverted and in the presence of other malicious parties, (ii) by running the protocol with the RF for honest implementation of P_i in the presence of other malicious parties. In [MS15], it was shown that for certain indistinguishability-based security notions like semantic security of an encryption scheme, an exfiltration resistant RF is also security preserving. It was postulated in [MS15] that, in general, when security requirements are simulation-based, ER *does not* imply SP. Surprisingly, we establish that exfiltration resistance *implies* security preservation for a reverse firewall when the security of the underlying protocol is simulation-based (computational) MPC security. For simulation-based security, this implication was only known for special functionalities like zero-knowledge. Our definitional implication shows that ER is the “right” notion for RF in the MPC setting; for new constructions, we need only construct RFs that are exfiltration resistant for each of the parties, and when all honest parties have an RF, security preservation for the protocol follows in the presence of malicious parties and arbitrary tampering of honest parties’ implementations. In the other direction, [MS15] showed that a security preserving RF is not necessarily ER when the underlying security does not promise privacy.

Reverse firewalls for adaptively secure MPC. The adaptive security notion for an MPC protocol models the realistic threat that an adversary can corrupt a party during the execution of a protocol. Adaptive security is much harder to achieve than static security for MPC. In the

reverse firewall setting, capturing a technical formulation of the adaptive security notion requires some care. When a party gets adaptively corrupted, the adversary can learn all of that party’s inputs and internal random coins. Consider an MPC protocol where an honest party deploys a firewall; now adaptively corrupting this party amounts to the adversary learning the composed state of the party with its reverse firewall. Typically, for reverse firewalls, security preservation means that the underlying security properties hold even under subversion. In the adaptive security case, we ask that adaptive security holds under subversion, where the adaptive adversary can learn the *composed state* of an adaptively corrupt party. Defining exfiltration resistance in the adaptive case needs some care. Here, we ask that the adversary not be able to distinguish between a tampered implementation of party P and an honest implementation, where the adversary can specify tampered implementations for initially honest parties and corrupt parties adaptively in the execution. While exfiltration resistance is not meaningful anymore once P gets corrupt in the middle of the protocol, our definition asks that up *until the point* that P gets corrupted, exfiltration resistance hold. Intuitively, the definition says that if P gets corrupted in the middle of execution, the adversary can see the composed state of P (the state of P composed with the state of the RF). Even given this state, the adversary should not be able to say if until corruption it was interacting with P composed with RF or \tilde{P} composed with RF, where \tilde{P} is a tampered implementation for P .

We construct reverse firewalls for *maliciously* secure MPC protocols in the presence of *adaptive* corruptions in the plain model. Similar to [CDN20], we consider RFs for functionality-maintaining tampering (see Sec. 4).

Theorem 1. (Informal) *Assuming DDH and LWE assumptions, there exists an $\mathcal{O}(1)$ round actively secure MPC protocol with reverse firewalls that is secure against adaptive corruptions in the urs model.*

Later, we generate (Thm. 3) the urs using an adaptively secure coin tossing protocol in the *plain* model based on the Discrete Logarithm (DLOG) and the Knowledge of Exponent (KEA) assumption in a different group. We consider this to be a result of independent interest, and further elaborate on the coin tossing protocol in the technical overview section.

Our approach is to construct an MPC protocol along the lines of GMW [GMW87], and add reverse firewall to this protocol. That is, our construction is essentially an *adaptive compiler*: it takes a semi-honest adaptively secure MPC protocol and runs [GMW87]-like steps in the reverse firewall setting to yield an adaptively secure MPC protocol with reverse firewalls. Towards this, we design adaptively secure protocols for augmented coin tossing and zero-knowledge, and construct reverse firewalls for each of the sub-protocols used in the compiler. Finally, we show that the compiled MPC protocol is adaptively secure in the presence of tampering of honest parties. We state each of the results below.

- *Reverse firewall for ZK*: Zero-knowledge in the presence of subversion have been studied in the form of parameter subversion for NIZK [BFS16], and in the RF setting for a class of interactive protocols called malleable sigma protocols [GMV20]. In this work, we consider interactive ZK since we aim for protocols without setup. Our protocol is a variant of the adaptively secure ZK protocol of [CSW20b] which is in the Uniform Random String (urs) model. Finally, we show how to design an RF for this protocol.

Theorem 2. (Informal) *Assuming LWE, there exists a three round actively secure ZK protocol with reverse firewalls that is secure against adaptive corruption of parties in the urs model.*

- *Reverse firewall for augmented coin-tossing*: We provide a construction of an adaptively-secure multi-party augmented coin-tossing protocol. Similar to our ZK protocol, our augmented coin-tossing protocol is also in the urs model. The main building block of our augmented

coin tossing protocol is an adaptively-secure commitment scheme (in the *urs* model) which is additively homomorphic over the message and randomness spaces. We then show how to construct an RF for this protocol.

Since our adaptively-secure augmented coin-tossing and ZK protocols are in the *urs* model, the compiled MPC protocol is also in the *urs* model. However, in the subversion setting we consider, a trusted setup is not available since a setup is susceptible to subversion too. For instance, the security guarantees of NIZKs completely break down in the face of subversion of the CRS [BFS16]. To circumvent the need for a trusted setup, we show how to generate the *urs* needed by our adaptively secure MPC protocol securely in the presence of subversion by presenting a multi-party coin tossing protocol with a reverse firewall in the plain model.

Adaptively secure coin tossing in plain model. As a contribution of independent interest, we construct an adaptively secure multi-party coin tossing protocol in the *plain model* under the knowledge of exponent (KEA) assumption. Our use of non-black-box assumptions seems justified, in light of the result of [GS12] that shows that it is not possible to construct an adaptively secure multi-party protocol with respect to black-box simulators without giving up on round efficiency in the plain model⁶. We use our coin-tossing protocol to generate the *urs* of our MPC protocol.

Theorem 3. (Informal) *Assuming DLOG, KEA and LWE assumptions, there exists a $\mathcal{O}(1)$ actively secure multi-party coin-tossing protocol that is secure against adaptive corruptions in the plain model.*

We then show how to add reverse firewalls to our adaptively secure coin tossing protocol.

Finally, putting everything together, we obtain an *adaptively* secure MPC protocol with reverse firewall in the *plain* model. This resolves the open question posed in [CDN20] of removing the trusted setup assumption in constructing MPC protocols with reverse firewalls.

1.2 Technical Overview

We provide a high-level overview of our construction, which can be viewed as an adaptive compiler for MPC protocols in the RF setting following the blueprint of [GMW87]. The main idea of the [GMW87] compiler is as follows: Each party (i) runs an instance of an augmented multi-party coin-tossing protocol to obtain a uniformly random string that it is committed to, (ii) commits to its input and broadcasts the input commitment to every other party, (iii) runs the underlying semi-honest adaptively secure MPC protocol, while proving in zero-knowledge that the computations have been done correctly. Since our goal is adaptive security, we start with an adaptively secure semi-honest protocol. Our compiler will use adaptively secure augmented coin-tossing and adaptively secure ZK protocols in the plain model.

Adding reverse firewalls. The protocol outlined above requires randomness in the augmented coin-tossing protocol and the ZK protocol. The rest of the MPC protocol is deterministic given the coins and the randomness of the ZK protocol. We propose an adaptively secure multi-party augmented coin-tossing protocol Π_{coin} and an adaptively secure (input-delayed) ZK protocol Π_{zk} . We then design reverse firewalls for these protocols and show that they provide exfiltration resistance for tampered parties. Then, by invoking our theorem that an exfiltration resistant RF is security preserving, we get that the RFs preserve security of the above protocols. We now explain them in more detail below.

⁶ If we had a coin tossing protocol with black-box simulation, we could use it to transform a two round adaptively secure MPC protocol in the URS model [CSW20a] to a protocol in the plain model by generating the URS via the coin toss protocol.

– $\Pi_{a\text{-coin}}$ using reverse firewalls: Our augmented coin-tossing uses the “commit-then-open” paradigm. At the end of this protocol, the initiating party (say, P_i) obtains a random string r_i along with the appropriate decommitment information, whereas all other parties $\{P_j\}_{j \in [n] \setminus i}$ obtain the (same) commitment to r_i . We assume that the message and randomness spaces of the commitment scheme form an additive group and the commitment scheme is *additively homomorphic* over these spaces. In the first round, each party $\{P_j\}_{j \in [n] \setminus i}$ sample their own randomness r_j and s_j , commits to the random coin r_j using s_j and broadcasts the commitment $c_j = \text{Com}(r_j; s_j)$. In the second round, party P_i samples its own randomness (r_i, s_i) , and broadcasts the commitment $c_i = \text{Com}(r_i; s_i)$ to all other parties. Finally, in the third round all parties $\{P_j\}_{j \in [n] \setminus i}$ broadcast their respective openings (r_j, s_j) . Party P_i then obtains the final string as $R = \sum_{k \in [n]} r_k$, and locally computes the commitment c_i as $c = \text{Com}(R; S)$, where $S = \sum_{k \in [n]} s_k$. All other parties can compute the same commitment c using the commitment c_i (broadcast by P_i) and the decommitment information of all other parties (broadcast in the final round) exploiting the homomorphic property of Com . We show that the above protocol is adaptively secure if the underlying commitment scheme Com is adaptively secure.

Consider the case when the initiating party P_i is tampered. In this case, the other malicious parties can launch an *input trigger attack* by sending a malformed commitment string which may serve as a wake up message to P_i . Besides, in the second round, tampered P_i can sample bad randomness and exfiltrate secrets via the commitment string c_i . When the receiving parties are corrupt, the commitment strings and their openings could also serve as a subliminal channel. The main idea of the RF is to exploit the homomorphic properties of Com to *sanitize* the incoming and outgoing messages. However, it must ensure that this mauling is consistent with the views of all parties. In particular, RF_i for P_i rerandomizes the commitment c_i to a fresh commitment \hat{c}_i by choosing fresh randomness (r'_i, s'_i) , computing $c'_i = \text{Com}(r'_i; s'_i)$ and homomorphically adding them. In the final round, when all the parties send their openings (r_j, s_j) , RF_i computes an additive secret sharing of r'_i and s'_i (sampled in the above step) and sanitizes each of these openings using the appropriate shares. Thus, the views of all the parties are consistent in this firewalled protocol and the final coin is also guaranteed to be random (since the offsets r'_i and s'_i were sampled randomly). Note that, the final commitment C computed by all the parties does not provide any channel to exfiltrate (since both R and S are random at the end of the firewalled execution). The detailed protocol together with the RF is in Section 7.1.

– Π_{zk} using reverse firewalls : Next, we need a ZK protocol to show conformance of each step of the protocol specification. We construct a reverse firewall for (a variant of) the adaptively secure ZK protocol of [CSW20b]. The protocol of [CSW20b] is based on the Sigma protocol of [FLS99] where the prover sends a first message, the verifier sends a random bit string as a challenge, and the prover sends a response in a third message. Towards constructing a reverse firewall, we observe that the prover’s messages can be re-randomized if the underlying primitives are homomorphic. However, the challenge string cannot be re-randomized, without also mauling the response provided by the prover. The ZK protocol of [CSW20b] does not seem to have this malleable property. Therefore, we modify the protocol, where the verifier’s challenge is generated as the result of a coin-tossing protocol. This ensures that the challenge is indeed random, and after the firewall sanitizes, both the prover and the verifier have the same challenge string. Therefore, the firewall can sanitize the protocol without the need to explicitly maul the response from the prover. The modified protocol remains adaptively secure. Note that the protocol also retains the input-delayed property – only the last round of the ZK protocol depends on the statement being proven and the corresponding witness. This allows running the first two rounds of the protocol before the inputs in the MPC protocol are defined. During the MPC protocol, the parties compute the input commitments and the protocol messages which define the statement and the witness. The last round of the ZK protocol is run after

this is defined, thus helping to preserve the round complexity of the underlying semi-honest adaptively-secure MPC protocol.

The idea behind a firewall for a tampered party in this modified ZK protocol, is to re-randomize the prover’s first message in the coin tossing homomorphically, thus ensuring that the verifier’s challenge in the sigma protocol is random. We show reverse firewalls for the prover and the verifier, and prove exfiltration resistance.

We obtain the above protocols in the urs model by instantiating the semi-honest MPC protocol and the underlying primitives in the urs model based on DDH and LWE (see Sections 7.1,7.2, and 7.3). Next, we generate the urs using an adaptively-secure coin tossing protocol to remove the setup assumption.

Adaptively secure coin tossing in the plain model. In order to remove the setup, we construct a constant round multi-party coin tossing protocol Π_{coin} in the plain model that generates the urs required by the MPC protocol. In addition to Discrete Log (DL) and Learning With Errors (LWE) assumption, we rely on knowledge of exponent assumption (KEA) assumption in pairing groups. Since it is *impossible* to construct a constant round adaptively secure coin-tossing protocol in the plain model from black-box simulation techniques [GS12], our reliance on the KEA assumption seems justified. The high-level idea behind our Π_{coin} protocol is as follows: There is an initial coin-tossing phase that sets up a public key of a homomorphic, obviously sampleable encryption scheme. In subsequent steps, there is another coin-tossing phase where parties exchange commitments to their coins together with encryption of the commitment randomness under the public key generated in the previous coin-toss. In more detail, the protocol uses the Pedersen commitment scheme – an *equivocal, perfectly hiding* commitment scheme, and a public key encryption scheme with additional properties. Π_{coin} consists of four phases - parameter generation phase, commitment generation phase, commitment opening phase and output phase.

In the first phase, the parties generate pairwise Pedersen commitment parameters and pairwise encryption key. For the commitment parameter, one party is the committer and the other party is the verifier; and the verifier additionally proves knowledge of the commitment trapdoor. The public key is of an encryption scheme that satisfies the following properties: oblivious ciphertext sampling, oblivious public key sampling and additive homomorphism of ciphertexts and public keys. The parameter generation is repeated by reversing the roles. In the commitment generation phase, each party generates its random coin and commits (as the committer) to it pairwise using the pairwise commitment parameters generated in the previous phase. In addition to the commitment, each party also sends two encryptions e_0 and e_1 : if the committed coin is $b \in \{0, 1\}$, then e_b is an encryption of the randomness used to commit to the coin, and e_{1-b} is sampled obliviously. Upon obtaining pairwise commitments to the random coins, the parties open their commitments pairwise by sending the decommitment randomness and encryption randomness for b to the pairwise verifiers. e_{1-b} is claimed to be obliviously sampled. Each party also broadcasts its random coin b . In the output phase, each party verifies the pairwise commitment openings and that correct ciphertext is an encryption of the commitment randomness. If all the openings are correct and they are consistent with the broadcasted coins then the parties output the final coin by summing up all the broadcasted coins.

This protocol is adaptively secure if the commitment is equivocal and perfectly hiding. The simulator needs to bias the output coin to a simulated coin. It is performed as follows: In the parameter generation phase, the verifier proves knowledge of trapdoor using a sub-protocol. When the verifier is corrupt, a non-black-box assumption allows extraction of the

trapdoor. When the committer is corrupt, the simulator receives the commitment and the public key, samples a key pair, rewinds the committer and sets its own oblivious key such that they homomorphically combine to the honestly sampled key. Now, the simulator knows the corresponding secret key. The simulator extracts the committed coins of the malicious parties in the commitment generation phase. In the opening phase the simulator equivocates (using the extracted trapdoors) the pairwise commitments and the coins broadcasted on behalf of the honest parties such that the final output coin is equal to the simulated output coin. Once the committer opens its public key in the first coin-toss, the simulator can rewind and force the output of this coin-toss phase to be a public key for which the simulator knows the secret key. In the subsequent coin-tossing phase where parties exchange commitment to their coins together with encryption of the commitment randomness, the simulator can extract the value committed. Crucially, the simulator can extract the committed coin of the corrupt committer *before* the adversary can see the output of the coin toss allowing it to simulate. When the committer is honest, the simulator can explain the ciphertexts as encrypting the correct values.

Adding reverse firewalls to the coin tossing protocol. We exploit the homomorphism property of the underlying commitment and public-key encryption scheme to sanitize round messages. In addition to this, the RF computes pairing equations in order to verify validity of messages.

On the setup assumption. The work of [CDN20] required a structured setup due its augmented coin-tossing protocol. In their coin tossing protocol the receiving parties obtain commitments to the sender’s coin, which is different from the commitment generated by the sender. As a result, during the later part of the protocol the RF needs to maul the proofs using a controlled-malleable NIZK (cm-NIZK) as the statement being proven by the sender is different from the one being verified by the receiving parties. Unfortunately, cm-NIZKs are not known in the adaptive setting. We modify the coin-tossing protocol such that every party obtains the *same* commitment string, and hence the proof statement remains unchanged. Thus, we can use an interactive ZK protocol without needing controlled malleability (re-randomizability suffices), and this allows us to rely on *urs* instead of *crs*. Finally, we can use the coin-tossing protocol (in the plain model) to remove the need for *urs*.

Finally, in all our protocols we rely on the existence of broadcast channels in the RF setting. We implicitly use the protocol of [CDN20], who showed how to implement broadcast channels in the RF setting.

1.3 Other Related Work

Besides the reverse firewall framework, other directions that address the challenge of protecting cryptosystems against different forms of subversion are reviewed below.

Algorithm Substitution Attacks. Bellare, Patterson, and Rogaway [BPR14] initiated the study of subversion of symmetric encryption schemes in the form of algorithm-substitution attacks (ASAs). They show that such subversion of the encryption algorithm is possible in a way that is undetectable. They also show that deterministic, stateful, ciphers are secure against this type of ASAs. Subsequent works redefined and strengthened the notion in several aspects [DFP15, BJK15], and extended the ASA model to other contexts, like digital signatures schemes [AMV15], public key encryption [CHY20].

Backdooring. Motivated by the backdooring of the DUAL EC DRBG [SF07], a formal study of backdooring of PRGs was initiated in [DGG⁺15], where public parameters are surreptitiously generated together with secret backdoors by a saboteur that allows to bypass security while remaining secure to any adversary that does not know the backdoor. Parameter subversion has been considered for several primitives, including pseudorandom generators [DGG⁺15, DPSW16], non-interactive zero knowledge [BFS16], and public-key encryption [ABK18].

Watchdogs and Self-guarding. Another approach taken in [RTYZ16, RTYZ17, BCJ21] is to consider an external entity called a watchdog that is trusted to test whether a given cryptographic implementation is compliant with its specification via black-box access. Self-guarding is another approach to counter subversion [FM18]. The idea here is to not depend on external entities, instead assume a trusted initialization phase where the cryptosystem is unsubverted.

2 Preliminaries

Notation. We write PPT to denote a probabilistic polynomial time machine. We denote the security parameter by λ . For an integer $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, 2, \dots, n\}$ and for any pair of integers $1 < i < j \leq n$, we denote by $[i, j]$ the set $\{i, i+1, \dots, j\}$. 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. We denote by \mathbb{G} the multiplicative group where DDH assumption holds. The corresponding field is denoted by \mathbb{Z}_q . We denote a negligible function in λ as $\text{neg}(\lambda)$.

2.1 Bilinear Groups and Knowledge of Exponent Assumption [AF07].

Our construction in the plain model is under Knowledge of Exponent Assumption. We present the definitions below.

Let \mathcal{BGG} denote a bilinear group generator. It takes in input the security parameter λ and outputs $(\mathbb{G}, \mathbb{H}, q, g, e)$ where \mathbb{G} and \mathbb{H} is a pair of groups of prime order q where g is a generator of group \mathbb{G} , and e is a non-degenerate bilinear map defined as $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{H}$ for which $e(g^a, g^b) = e(g, g)^{ab}$ for $a, b \in \mathbb{Z}_q$ and $e(g, g) \neq 1_{\mathbb{H}}$.

Definition 1. (Discrete Log Assumption) For every non-uniform poly-time algorithm \mathcal{A} :

$$\Pr[\text{pub} \leftarrow \mathcal{BGG}(1^\lambda), h \leftarrow \mathbb{G}, w \leftarrow \mathcal{A}(\text{pub}, h) : g^w = h] \leq \text{neg}(\lambda)$$

Definition 2. (Knowledge of Exponent Assumption). For every non uniform poly-time algorithm \mathcal{A} there exists a non-uniform poly-time algorithm $\mathcal{X}_{\mathcal{A}}$, the extractor such that:

$$\Pr[\text{pub} \leftarrow \mathcal{BGG}(1^\lambda), x \leftarrow \mathbb{Z}_q, (A, \hat{A}; a) \leftarrow (\mathcal{A} || \mathcal{X}_{\mathcal{A}})(\text{pub}, g^x) :$$

$$\hat{A} = A^x \wedge A \neq g^a] \leq \text{neg}(\lambda)$$

where $(A, \hat{A}; a) \leftarrow (\mathcal{A} || \mathcal{X}_{\mathcal{A}})(\text{pub}, g^x)$ means that \mathcal{A} and $\mathcal{X}_{\mathcal{A}}$ are executed on the same input (pub, g^x) and the same random tape, and \mathcal{A} outputs (A, \hat{A}) whereas $\mathcal{X}_{\mathcal{A}}$ outputs a .

2.2 Public Key Encryption Schemes

A public key encryption scheme $\text{PKE} = (\text{Gen}, \text{Enc}, \text{Dec})$ satisfies *oblivious ciphertext sampling* if there exists a polynomial time algorithm oEnc which obviously samples a ciphertext s.t. it looks indistinguishable from a real ciphertext. Additionally, we require the PKE to satisfy *additive homomorphism* over message and randomness space, i.e. $\text{Enc}(\text{pk}, m; r) \cdot \text{Enc}(\text{pk}, m'; r') = \text{Enc}(\text{pk}, m + m'; r + r')$.

Definition 3. ([CSW20a] (Public Key Encryption with oblivious ciphertext sampling and oblivious public keys sampling)) A public key encryption scheme $\text{PKE} = (\text{Gen}, \text{Enc}, \text{Dec})$ over message space \mathcal{M} , ciphertext space \mathcal{C} and randomness space \mathcal{R} satisfies *oblivious ciphertext sampling property* if there exists PPT algorithms oGen, oEnc s.t. the following holds:

- *Oblivious Ciphertext Sampling*: For any message $m \in \mathcal{M}$, the following two distributions are computationally indistinguishable to a PPT adversary \mathcal{A} :

$$\left| \Pr[\mathcal{A}(m, c) = 1 \mid (\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^\lambda), m \leftarrow \mathcal{A}(\text{pk}), c \leftarrow \text{Enc}(\text{pk}, m)] \right.$$

$$\left. - \Pr[\mathcal{A}(m, \tilde{c}) = 1 \mid (\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^\lambda), m \leftarrow \mathcal{A}(\text{pk}), \tilde{c} \leftarrow \text{oEnc}(\text{pk})] \right| \leq \text{neg}(\lambda)$$

- *Oblivious Public Key Sampling*: The following two distributions are computationally indistinguishable to a PPT adversary \mathcal{A} :

$$\left| \Pr[\mathcal{A}(\text{pk}) = 1 \mid (\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^\lambda)] - \Pr[\mathcal{A}(\text{pk}') = 1 \mid \text{pk}' \leftarrow \text{oGen}(1^\lambda)] \right| \leq \text{neg}(\lambda)$$

Definition 4. (Additively Homomorphic Encryption) A public key encryption scheme $\text{PKE} = (\text{Gen}, \text{Enc}, \text{Dec})$ over message space \mathcal{M}_{Enc} and randomness space \mathcal{R}_{Enc} is additively homomorphic over message space \mathcal{M}_{Enc} and randomness space \mathcal{R}_{Enc} , which are written additively, such that for all $m, m' \in \mathcal{M}_{\text{Enc}}, r, r' \in \mathcal{R}_{\text{Enc}}$ we have:

$$\text{Enc}(\text{pk}, m; r) \cdot \text{Enc}(\text{pk}, m'; r') = \text{Enc}(\text{pk}, m + m'; r + r')$$

where $(\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^\lambda)$.

2.3 Commitment Schemes

We denote a commitment scheme as $\text{Com} = (\text{Gen}, \text{Com}, \text{Verify})$. It is *equivocal* if there exists a polynomial time algorithm Equiv that equivocates a commitment to open to any message, given the trapdoor of the commitment parameters. We also need an adaptively secure commitment scheme and we use the definition of [CSW20a]. We also need the commitment scheme to satisfy additively homomorphic property like the PKE scheme.

We present the ideal commitment functionality from [CSW20a] in Fig. 1. We define a non-interactive perfectly equivocal commitment scheme $\text{Com} = (\text{Gen}, \text{Com}, \text{Verify}, \text{Equiv})$ as follows:

Fig. 1. The ideal functionality \mathcal{F}_{Com} for Commitment Scheme

\mathcal{F}_{Com}
\mathcal{F}_{Com} interacts with committer C and verifier V as follows: <ul style="list-style-type: none"> – On receiving input $((\text{Commit}, \text{V}), \text{C}, \text{sid}, m)$ from C, if $(\text{sid}, \text{C}, \text{V}, m')$ has been recorded, ignore the input. Else record the tuple $(\text{sid}, \text{C}, \text{V}, m)$ and send $(\text{RECEIPT}, \text{sid}, \text{C}, \text{V})$ to V. – On receiving input $(\text{Open}, \text{C}, \text{sid})$ from C, if there is a record of the form $(\text{sid}, \text{C}, \text{V}, m')$ return $(\text{Open}, \text{sid}, \text{C}, \text{V}, m')$ to V. Otherwise, ignore the input.

Definition 5. (Correctness) Com is a correct commitment scheme if the following holds true

$$\Pr[\text{Verify}(\text{pp}, c, m, r) = 1 \mid (\text{pp}, \text{td}) \leftarrow \text{Gen}(1^\lambda), c \leftarrow \text{Com}(\text{pp}, m; r)] = 1$$

Definition 6. (Computationally Binding) Com is computationally binding scheme if the following holds true for all PPT adversary \mathcal{A}

$$\Pr[(m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(\text{pp}) \mid (\text{pp}, \text{td}) \leftarrow \text{Gen}(1^\lambda),$$

$$\text{Com}(\text{pp}, m_0; r_0) = \text{Com}(\text{pp}, m_1; r_1)] \leq \text{neg}(\lambda)$$

Definition 7. (Perfect Binding) Com is perfectly binding scheme if the following holds true for all unbounded adversary \mathcal{A}

$$\Pr [(m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(\text{pp}) | (\text{pp}, \text{td}) \leftarrow \text{Gen}(1^\lambda), \\ \text{Com}(\text{pp}, m_0; r_0) = \text{Com}(\text{pp}, m_1; r_1)] = 0$$

Definition 8. (Computationally Hiding) Com is a computationally hiding scheme if the following holds true for all PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

$$\Pr [b = b' | (\text{pp}, \text{td}) \leftarrow \text{Gen}(1^\lambda), (m_0, m_1, \text{st}) \leftarrow \mathcal{A}_1(\text{pp}), b \leftarrow \{0, 1\}, \\ c \leftarrow \text{Com}(\text{pp}, m_b; r), b' \leftarrow \mathcal{A}_2(c; \text{st})] \leq \frac{1}{2} + \text{neg}(\lambda)$$

Definition 9. (Perfectly Hiding) Com is perfectly hiding scheme if the following holds true for all computationally unbounded adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

$$\Pr [b = b' | (\text{pp}, \text{td}) \leftarrow \text{Gen}(1^\lambda), (m_0, m_1, \text{st}) \leftarrow \mathcal{A}_1(\text{pp}), b \leftarrow \{0, 1\}, \\ c \leftarrow \text{Com}(\text{pp}, m_b; r), b' \leftarrow \mathcal{A}_2(c; \text{st})] = \frac{1}{2}$$

Definition 10. (Perfectly Equivocal) Com is equivocal if it has a PPT algorithm \equiv s.t. the following holds true for all computationally unbounded adversary \mathcal{A} and all message pairs (m, m') for $m \neq m'$.

$$\left| \Pr [\mathcal{A}(c, r) = 1 | (\text{pp}, \text{td}) \leftarrow \text{Gen}(1^\lambda), (m, m') \leftarrow \mathcal{A}(\text{pp}), c = \text{Com}(\text{pp}, m; r)] \right. \\ \left. - \Pr [\mathcal{A}(c, r) = 1 | (\text{pp}, \text{td}) \leftarrow \text{Gen}(1^\lambda), (m, m') \leftarrow \mathcal{A}(\text{pp}), c = \text{Com}(\text{pp}, m'; r'), \right. \\ \left. r = \text{Equiv}(\text{pp}, c, m, m', r', \text{td}) \right] = 0$$

Definition 11. (Additively Homomorphic Commitment) A commitment scheme $\text{Com} = (\text{Gen}, \text{Com}, \text{Verify})$ is additively homomorphic over message space \mathcal{M}_{Com} and randomness space \mathcal{R}_{Com} , which are written additively, such that for all $m, m' \in \mathcal{M}_{\text{Com}}$, $r, r' \in \mathcal{R}_{\text{Com}}$ we have:

$$\text{Com}(\text{pp}, m; r) \cdot \text{Com}(\text{pp}, m'; r') = \text{Com}(\text{pp}, m + m'; r + r')$$

where $(\text{pp}, \text{td}) \leftarrow \text{Gen}(1^\lambda)$.

Pedersen Commitment Scheme. Given a generator $g, h \in \mathbb{G}$, the committer commits to a field element $m \leftarrow \mathbb{Z}_q$ by sampling randomness $r \leftarrow \mathbb{Z}_q$ and sets $c = g^m h^r$. Decommitment to message m is r . It is perfectly hiding and computationally binding due to the Discrete Log assumption.

We present a version of Elgamal commitment scheme without setup as follows. Given a generator $g \in \mathbb{G}$, the committer commits to a field element $m \leftarrow \mathbb{Z}_q$ by sampling randomness $x, r \leftarrow \mathbb{Z}_q$ and sets $c = (c_1, c_2, c_3) = (g^x, g^r, g^m g^{rx}) = (h, g^r, g^m h^r)$. The tuple (x, r) serves as the decommitment information. It is perfectly binding and computationally hiding due to the DDH assumption.

Fig. 2. Zero-Knowledge Functionality \mathcal{F}_{ZK} (from [CSW20b]) for a single prover proving multiple statements

\mathcal{F}_{ZK} is parametrized by an NP relation \mathcal{R} .

- On input (prove, sid, x, w) from P and (verify, sid, x) from V : if there exists $(\text{sid}, P') \in \mathcal{Q}$ and $P \neq P'$ or $\mathcal{R}(x, w) \neq 1$ then ignore the input. Else record $\mathcal{Q} = (\text{sid}, P)$ and output (verification, sid, $x, \mathcal{R}(x, w)$) to V.

2.4 Zero Knowledge

The prover proves that a statement $x \in \mathcal{L}$ by interacting with a verifier V. The prover has the knowledge of a secret witness w s.t. $\mathcal{R}(x, w) = 1$ iff $x \in \mathcal{L}$. An interactive ZK protocol consists of $\Pi_{\text{ZK}} = (\text{Gen}, \text{P}, \text{V})$ where Gen generates the urs and P and V are interactive algorithms. We present the ZK functionality from [CSW20b] in Fig. 2. It also allows the prover to prove multiple statements to a verifier using the same setup string. If a protocol Π_{ZK} implements \mathcal{F}_{ZK} then there exists a simulator who can extract correct witnesses from the accepting proofs when the prover is corrupt.

We also require the ZK protocols to be input-delayed, i.e. only the last message from the prover to the verifier should depend on the statement. We denote by $\langle \text{P}(w), \text{V} \rangle(x, \text{urs})$ the distribution of V's output after running Π_{ZK} with P on public input (x, urs) and witness w . Input delayedness is defined as follows.

Definition 12. (Input-Delayed ZK protocols) Let $\Pi_{\text{ZK}} = (\text{Gen}, \text{P}, \text{V})$ is a $2r + 1$ -round interactive protocol that implements \mathcal{F}_{ZK} (Fig. 2) functionality where P sends the first and last round message. We denote $\text{P} = \{\text{P}_i\}_{i \in [r+1]}$ where P_i denotes the prover algorithm for computing $(2(i - 1) + 1)$ -th round message of Π_{ZK} . Similarly, $\text{V} = (\{\text{V}_i\}_{i \in [r]}, \text{V}^{\text{out}})$ where V_i denotes the verifier algorithm for computing $2i$ -th round message of Π_{ZK} and V^{out} either accepts or rejects the proof. Π_{ZK} is input-delayed ZK protocol if the following holds:

- $\{\text{P}_i\}_{i \in [r]}$ takes as input the private state of P_{i-1} and the public values - length of the statement, i.e. $|x|$, urs and previous round messages.
- $\{\text{V}_i\}_{i \in [r]}$ takes as input the private state of V_{i-1} and the public values - length of the statement, i.e. $|x|$, urs and previous round messages.
- P_{r+1} takes as input (x, w) and private state of P_r .
- V^{out} takes as input x and private state of V_r .

2.5 Coin-Tossing

Definition 13 ([GMW87] Multi-party 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 functionality $(1^\lambda, \dots, 1^\lambda) \rightarrow (U_t, U_t, \dots, U_t)$, where U_t denotes the uniform distribution over t -bit strings.

Definition 14 ([GMW87] 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 functionality $(1^\lambda, \dots, 1^\lambda) \rightarrow ((U_t, U_{t,\lambda}),$

$\text{Com}(U_t; U_{t,\lambda}), \dots, \text{Com}(U_t; U_{t,\lambda}))$ with respect to a fixed commitment scheme $\text{Com} = (\text{Gen}, \text{Com}, \text{Verify})$ which requires λ random bits to commit to each bit, and U_t denotes the uniform distribution over t -bit strings.

2.6 Adaptively Secure Multi-Party Computation

In this section we recall the formal definition of adaptively secure MPC protocols in the *stand-alone* setting as in [GS12].

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 cast by specifying a n -ary *functionality*, denoted by $f : (\{0, 1\}^*)^n \rightarrow (\{0, 1\}^*)^n$, where $f = (f_1, \dots, f_n)$. For an input vector $\vec{x} = \{x_1, \dots, x_n\}$ the output is a tuple of random variables denoted by $(f_1(\vec{x}), \dots, f_n(\vec{x}))$. The i^{th} party P_i initially holds the input x_i and obtains $f_i(\vec{x})$. We also assume that all the parties hold input of equal length, i.e., $|x_i| = |x_j|$ for all $i, j \in [n]$.

Adversarial behavior. For the analysis of our protocols we consider the setting of *malicious* adversaries that can *adaptively* corrupt parties throughout the protocol execution depending on its view during the execution. We consider the definition of security in terms of the real-world and ideal-world simulation paradigm.

Real World. In the real world, the MPC protocol Π is executed by the interaction of n parties $\{P_1, \dots, P_n\}$. Each party P_i has input $x_i \in \{0, 1\}^*$, random input $r_i \in \{0, 1\}^*$, and the security parameter λ . Let $C \subset [n]$ and $H = [n] \setminus C$ denote the indices of the malicious corrupted parties and honest parties in Π . Consequently, let us denote by P_C and P_H the set of maliciously corrupted and honest parties respectively. We assume that all communication is done via a *broadcast* channel. We consider the synchronous, with rushing model of computation.

At the onset of the computation the adversary \mathcal{A} receives some auxiliary input denoted by z . The computation proceeds in *rounds*, with each round consisting of several *mini-rounds*. Each mini-round starts by allowing \mathcal{A} to *adaptively* corrupt parties one by one. Once a party is corrupted the party's input and random input become known to \mathcal{A} . Next, \mathcal{A} activates an uncorrupted party P_i , which has not been activated so far in this round. Upon activation, P_i receives the messages sent to it in the previous round, generates the message for this round, and the next mini-round begins. \mathcal{A} also gets to learn the messages sent by P_i . Once all the uncorrupted parties are activated, \mathcal{A} sends the messages on behalf of the corrupt parties that were not yet activated in this round, and the next round begins. Finally, at the end of the computation (after some pre-specified number of rounds) the parties locally generate their outputs. Each uncorrupted/honest parties output what is specified as in the protocol. The corrupt parties may output an arbitrary probabilistic polynomial-time function of the view of \mathcal{A} .

The overall output of the real-world experiment consists of the output of all parties at the end of the protocol, and the real world adversary view is denoted by $\text{REAL}_{\Pi, (C, \mathcal{A})}(\lambda, \vec{x}, \vec{r}, z)$. Let $\text{REAL}_{\Pi, (C, \mathcal{A})}(\lambda, \vec{x}, z)$ be the distribution of $\text{REAL}_{\Pi, (C, \mathcal{A})}(\lambda, \vec{x}, \vec{r}, z)$ when \vec{r} is chosen uniformly at random. Let $\text{REAL}_{\Pi, (C, \mathcal{A})}$ denote the distribution ensemble $\{\text{REAL}_{\Pi, (C, \mathcal{A})}(\lambda, \vec{x}, z)\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0, 1\}^*)^n, z \in \{0, 1\}^*}$.

Ideal World. In the ideal world we assume the existence of an incorruptible trusted third party (TTP), with whom all the parties interact. Each party P_i gets input $x_i \in \{0, 1\}^*$ and wish to evaluate $f_1(\vec{x}, r_f), \dots, f_n(\vec{x}, r_f)$, where $r_f \leftarrow \{0, 1\}^s$, and s is a value determined by the security parameter, and P_i learns $f_i(\vec{x}, r_f)$. The ideal world computation in the presence of an adaptive ideal world adversary Sim (with random input r) and the TTP TTP proceeds as in Fig. 3.

The overall output of the ideal-world experiment consists of the output of all parties at the end of the protocol, and the ideal world adversary view is denoted by $\text{IDEAL}_{f, (C, \text{Sim})}(\lambda, \vec{x}, \vec{r}, z)$, where $\vec{r} = (r, r_f)$. Let $\text{IDEAL}_{f, (C, \text{Sim})}(\lambda, \vec{x}, z)$ be the denote the distribution of $\text{IDEAL}_{f, (C, \text{Sim})}(\lambda, \vec{x}, \vec{r}, z)$ where \vec{r} is chosen uniformly at random. Also, let $\text{IDEAL}_{f, (C, \text{Sim})}$ denote the distribution ensemble $\{\text{IDEAL}_{f, (C, \text{Sim})}(\lambda, \vec{x}, z)\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0, 1\}^*)^n, z \in \{0, 1\}^*}$.

Now that the ideal and real world executions are defined, we put forward the notion of security for an adaptively secure multi-party protocol Π . Informally, we require that executing a protocol Π in the real world emulates the ideal process for evaluating f .

Definition 15 (Adaptive Security). Let f be any adaptive well-formed n -ary function, Π be a protocol for n parties. We say that Π adaptively securely evaluates f if for every real world adversary \mathcal{A} there exists an ideal world adversary Sim , such that $\text{REAL}_{\Pi,(\mathcal{C},\mathcal{A})} \approx_c \text{IDEAL}_{f,(\mathcal{C},\text{Sim})}$.

Fig. 3. The Ideal World Execution of an Adaptive MPC Protocol

- **Input:** Let $\vec{x} = \{x_1, \dots, x_n\}$ denote the initial inputs for the parties. As in the real-world, the adversary/simulator Sim additionally has an auxiliary input denoted by z .
- **First corruption stage:** Sim proceeds in iterations, where in each iteration Sim may decide to corrupt some party based on the random input of Sim . Once a party is corrupted its input becomes known to Sim .
- **Send inputs to TTP:** Each honest party P_i sends its input x_i to TTP. The corrupted parties may either abort (send special symbol \perp), send correct input x_i , or send some other input x'_i ($|x_i| = |x'_i|$) to TTP. Let $\vec{x}' = \{x'_1, \dots, x'_n\}$ denote the input vector received by TTP.
- **Early abort option:** If TTP receives the special symbol \perp , it sends abort to the honest parties and the ideal execution terminates. Otherwise,
- **TTP sends output to adversary:** TTP chooses r_f uniformly at random, computes $f(\vec{x}', r_f)$ and sends it to the Sim first.
- **Adversary Sim instructs TTP to continue or halt:** Sim either sends continue or abort to TTP. In case of continue, TTP sends $f(\vec{x}', r_f)$ to the honest parties. Otherwise, if Sim sends abort, TTP sends abort to the honest parties.
- **Second corruption stage:** Upon learning the corrupted parties' outputs of the computation, Sim proceeds in another sequence of iterations, where in each iteration Sim may decide to corrupt some additional parties, based on the information gathered so far. Upon corruption, Sim learns the sees the corrupted party's input and output.
- **Output stage:** Each honest party output the output received from TTP, while the maliciously corrupted parties P_C output any probabilistic polynomial-time computable function of their input, the auxiliary input z , and the output received from TTP.
- **Post-execution corruption:** Once the outputs are generated, Sim may at any point in the protocol may decide to adaptively proceed in another sequence of iterations, where in each iteration Sim may decide to corrupt some additional party, based on the information gathered so far.

In this section, we recall RF definitions in the static case [CDN20],[MS15].

3 Reverse Firewalls for Statically secure MPC

Definition 16. (Exfiltration-resistant RF in the presence of static corruptions). Let Π be a multi-party protocol run between the parties P_1, \dots, P_n satisfying functionality \mathcal{F} and having reverse firewalls RF_i for the set of honest parties $\{P_i\}_{i \in \mathcal{H}}$. Then $\forall i \in \mathcal{H}$, we say that the firewall RF_i is exfiltration-resistant for party P_i against all other parties $\{P_j\}_{j \in [n] \setminus i}$, if for any PPT adversary \mathcal{A}_{ER} , the advantage $\text{Adv}_{\mathcal{A}_{\text{ER}}, \text{RF}_i}^{\text{LEAK}}(\lambda)$ of \mathcal{A}_{ER} (defined below) in the game **LEAK** (see Fig. 4) is negligible in the security parameter λ .

The advantage of any adversary \mathcal{A}_{ER} in the game **LEAK** is defined as:

$$\text{Adv}_{\mathcal{A}_{\text{ER}}, \text{RF}_i}^{\text{LEAK}}(\lambda) = \left| \Pr[\text{LEAK}(\Pi, i, \{P_1, \dots, P_n\}, \text{RF}_i, \lambda) = 1] - \frac{1}{2} \right|.$$

Definition 17. (Security-preserving RF for Malicious Statically secure MPCs). Let Π be a multi-party protocol run between the parties P_1, \dots, P_n satisfying functionality requirement \mathcal{F} and is secure against adaptive malicious adversaries (see Def. 15) as above. We assume that each honest party $\{P_i\}_{i \in \mathcal{H}}$ is equipped with its corresponding reverse firewall $\{\text{RF}_i\}_{i \in \mathcal{H}}$. Then, we say that the reverse firewalls RF_i for parties $\{P_i\}_{i \in \mathcal{H}}$ strongly (resp. weakly) preserves security of the protocol Π , 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 world into polynomial-size circuit families $\text{Sim} = \{\text{Sim}_\lambda\}_{\lambda \in \mathbb{N}}$ for the ideal model such that for every $\lambda \in \mathbb{N}$, every subset $\mathcal{H} \subset [n]$, every input sequence $\vec{x} = (x_1, \dots, x_n) \in (\{0, 1\}^\lambda)^n$, every auxiliary information $z \in \{0, 1\}^*$ and every arbitrary (resp. functionality-maintaining) tampered implementation $\{\overline{P}_i\}_{i \in \mathcal{H}}$ we have that:

Fig. 4. $\text{LEAK}(\Pi, i, \{P_1, \dots, P_n\}, \text{RF}_i, \lambda)$ is the exfiltration-resistance security game for reverse firewall RF_i for party P_i in the protocol Π against other parties $\{P_j\}_{j \in [n \setminus i]}$ with input I . Here, $\{\text{st}_{P_j}\}_{j \in [n \setminus i]}$ denote the states of the parties $\{P_j\}_{j \in [n \setminus i]}$ after the run of the protocol, and \mathcal{T}^* denote the transcript of the protocol $\Pi_{P_i \rightarrow P_i^*, \{P_j \rightarrow \overline{P_j}\}_{j \in [n \setminus i]}}(I)$.

$$\begin{array}{c}
 \text{LEAK}(\Pi, i, \{P_1, \dots, P_n\}, \text{RF}_i, \lambda) \\
 (\overline{P}_1, \dots, \overline{P}_n, I) \leftarrow \mathcal{A}_{\text{ER}}(1^\lambda) \\
 b \xleftarrow{\$} \{0, 1\}; \\
 \text{If } b = 1, P_i^* \leftarrow \text{RF}_i \circ \overline{P}_i \\
 \text{Else, } P_i^* \leftarrow \text{RF}_i \circ P_i. \\
 \mathcal{T}^* \leftarrow \Pi_{\{P_i^*, \{P_j \rightarrow \overline{P_j}\}_{j \in [n \setminus i]}\}}(I). \\
 b^* \leftarrow \mathcal{A}_{\text{ER}}(\mathcal{T}^*, \{\text{st}_{P_j}\}_{j \in [n \setminus i]}). \\
 \text{Output } (b = b^*).
 \end{array}$$

$$\text{REAL}_{\Pi_{\{\text{RF}_i \circ \overline{P}_i\}_{i \in \text{H}}, (\text{C}, \mathcal{A})}}(\lambda, \vec{x}, z) \approx_{\text{c}} \text{IDEAL}_{f, (\text{C}, \text{Sim})}(\lambda, \vec{x}, z).$$

Definition 18. (Transparent RF in presence of static corruptions). Let Π be a multi-party protocol run between the parties P_1, \dots, P_n satisfying functionality \mathcal{F} and having reverse firewalls RF_i for the set of honest parties $\{P_i\}_{i \in \text{H}}$. Then $\forall i \in \text{H}$, we say that the firewall RF_i is transparent for party P_i against all other parties $\{P_j\}_{j \in [n \setminus i]}$, if for any PPT adversary \mathcal{A}_{tr} , the advantage $\text{Adv}_{\mathcal{A}_{\text{tr}}, \text{RF}_i}^{\text{TRANS}}(\lambda)$ of \mathcal{A}_{tr} (defined below) in the game **TRANS** (see Fig. 5) is negligible in the security parameter λ .

The advantage of any adversary \mathcal{A}_{tr} in the game **TRANS** is defined as:

$$\text{Adv}_{\mathcal{A}_{\text{tr}}, \text{RF}_i}^{\text{TRANS}}(\lambda) = \left| \Pr[\text{TRANS}(\Pi, i, \{P_1, \dots, P_n\}, \text{RF}_i, \lambda, \text{C}) = 1] - \frac{1}{2} \right|.$$

Fig. 5. The transparency game for a reverse firewall RF_i for a party P_i for adaptively secure MPCs. Here, C denote the set of corrupt parties.

$$\begin{array}{c}
 \text{TRANS}(\Pi, i, \{P_1, \dots, P_n\}, \text{RF}_i, \lambda, \text{C}) \\
 (I) \leftarrow \mathcal{A}_{\text{ER}}(1^\lambda) \\
 b \xleftarrow{\$} \{0, 1\}; \\
 \text{If } b = 1, P_i^* \leftarrow \text{RF}_i \circ P_i \\
 \text{Else, } P_i^* \leftarrow P_i. \\
 \mathcal{T}^* \leftarrow \Pi_{\{P_i^*, P_j \in \text{H}\}}(I). \\
 b^* \leftarrow \mathcal{A}_{\text{ER}}(\mathcal{T}^*, \{\text{st}_{P_j}\}_{j \in \text{C}}). \\
 \text{Output } (b = b^*).
 \end{array}$$

4 Reverse Firewalls for Adaptively secure MPCs

In this section, we present definitions of reverse firewalls for adaptively secure MPC protocols. The existing definitions of security preservation and exfiltration-resistance for reverse firewalls are for a static adversary [CDN20]. In the adaptive setting, while the security preservation is defined as before, exfiltration resistance now has to incorporate the adaptive power of the adversary. We also present a definition of transparency for reverse firewalls. We first introduce some notation that will be used throughout the paper.

Notation. Let Π denote a ℓ -round MPC protocol, for some arbitrary polynomial $\ell(\cdot)$ in the security parameter λ . Let H and C denote the indices of the honest and maliciously corrupted parties respectively in the protocol Π . For a party P and reverse firewall RF we define $\text{RF} \circ P$ as the “composed” party in which the incoming and outgoing messages of A are “sanitized” by RF . The firewall RF is a *stateful* algorithm that is only allowed to see the public parameters of the system, and does not get to see the inputs and outputs of the party P . We denote the tampered implementation of a party P by \overline{P} .

We denote the view of a party P_i by View_{P_i} , which consists of the input of P_i , its random tape and the messages received so far. We also denote the view of a party P_i till some round $k(\leq \ell)$ as $\text{View}_{P_i}^{\leq k}$. We denote the reverse firewall for party P_i as RF_i and the internal state of RF_i by st_{RF_i} . We write $\text{View}_{\text{RF}_i \circ P_i}$ to denote the composed view of a party P_i and its RF RF_i . Let $\text{Transform}(\cdot)$ be a polynomial time algorithm that takes as input the random tape r_i of a party P_i and the internal state (or randomness) st_{RF_i} of RF_i and returns a sanitized random tape $\text{Transform}(r_i, \text{st}_{\text{RF}_i})$ ⁷. Note that, the composed view $\text{View}_{\text{RF}_i \circ P_i}$ of P_i can be efficiently constructed from the view View_{P_i} of P_i and the state st_{RF_i} of RF_i using the Transform function as a subroutine. We write $\Pi_{\text{RF}_i \circ P_i}$ (resp. $\Pi_{\overline{P_i}}$) to represent the protocol Π in which the role of a party P_i is replaced by the composed party $\text{RF}_i \circ P_i$ (resp. the tampered implementation $\overline{P_i}$).

Definition 19. (Functionality-maintaining RF). For any reverse firewall RF and a party P , let $\text{RF}^1 \circ P = \text{RF} \circ P$, and $\text{RF}^k \circ P = \underbrace{\text{RF} \circ \dots \circ \text{RF}}_{k \text{ times}} \circ P$. For a protocol Π that satisfies some functionality requirements \mathcal{F} , we say that a reverse firewall RF maintains functionality \mathcal{F} for a party P in protocol Π if $\Pi_{\text{RF}^k \circ P}$ also satisfies \mathcal{F} , for any polynomially bounded $k \geq 1$.

Definition 20. (Security-preserving RF for Malicious Adaptively secure MPCs). Let Π be a multi-party protocol run between parties P_1, \dots, P_n satisfying functionality requirement \mathcal{F} and is secure against adaptive malicious adversaries (see Def. 15). We assume that each honest party $\{P_i\}_{i \in \mathcal{H}}$ is equipped with its corresponding reverse firewall $\{\text{RF}_i\}_{i \in \mathcal{H}}$. When the adversary (adaptively) corrupts a party P_i , it receives $\text{View}_{\text{RF}_i \circ P_i}$ as the view of P_i . Then, we say that the reverse firewalls RF_i for parties $\{P_i\}_{i \in \mathcal{H}}$ strongly (resp. weakly) preserves security of the protocol Π , 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 world into polynomial-size circuit families $\text{Sim} = \{\text{Sim}_\lambda\}_{\lambda \in \mathbb{N}}$ for the ideal model such that for every $\lambda \in \mathbb{N}$, every subset $\mathcal{H} \subset [n]$, every input sequence $\vec{x} = (x_1, \dots, x_n) \in (\{0, 1\}^\lambda)^n$, every auxiliary information $z \in \{0, 1\}^*$ and every arbitrary (resp. functionality-maintaining) tampered implementation $\{\overline{P_i}\}_{i \in \mathcal{H}}$ we have the following:

$$\text{REAL}_{\Pi_{\{\text{RF}_i \circ \overline{P_i}\}_{i \in \mathcal{H}}}, (\mathcal{C}, \mathcal{A})}(\lambda, \vec{x}, z) \approx_c \text{IDEAL}_{f, (\mathcal{C}, \text{Sim})}(\lambda, \vec{x}, z).$$

We now define exfiltration resistance in terms of the game LEAK that asks the adversary to distinguish between a tampered implementation of party P_i and an honest implementation, even given the composed state of P_i and its RF if P_i gets adaptively corrupt in the middle of execution.⁸

Definition 21. (Exfiltration-resistant RF in the presence of adaptive corruptions). Let Π be a multi-party protocol run between the parties P_1, \dots, P_n satisfying functionality \mathcal{F} and having reverse firewalls RF_i for the set of honest parties $\{P_i\}_{i \in \mathcal{H}}$. When the adversary (adaptively) corrupts a party P_i , it receives $\text{View}_{\text{RF}_i \circ P_i}$ as the view of P_i . Then $\forall i \in \mathcal{H}$, we say that the firewall RF_i is exfiltration-resistant for party P_i against all other parties $\{P_j\}_{j \in [n] \setminus i}$, if for any PPT adversary \mathcal{A}_{ER} , the advantage $\text{Adv}_{\mathcal{A}_{\text{ER}}, \text{RF}_i}^{\text{LEAK}}(\lambda)$ of \mathcal{A}_{ER} (defined below) in the game LEAK (see Fig. 6) is negligible in the security parameter λ .

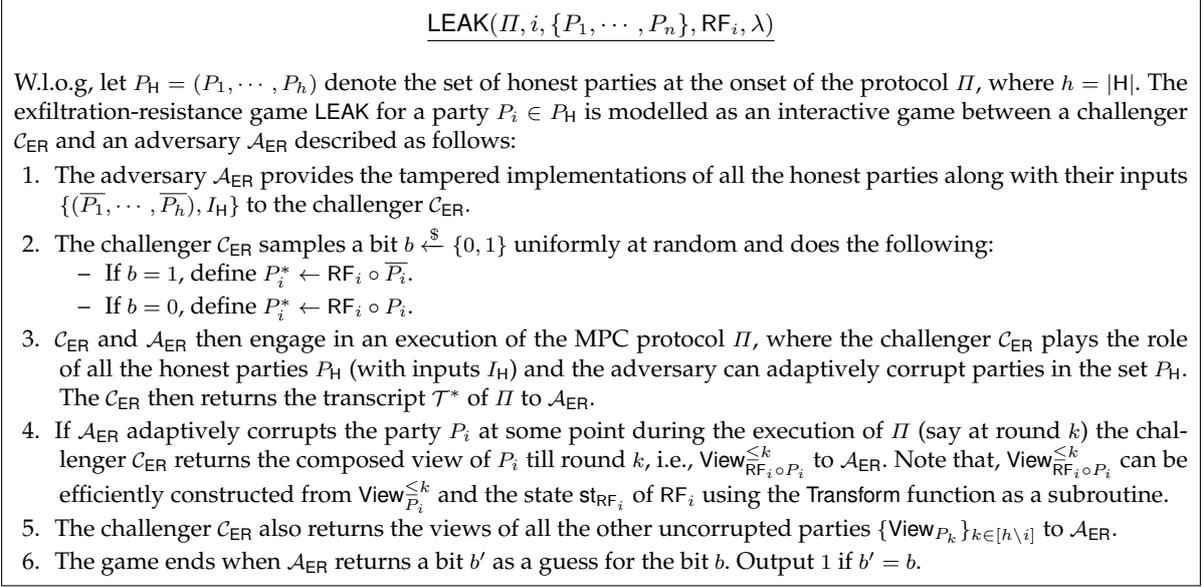
The advantage of any adversary \mathcal{A}_{ER} in the game LEAK is defined as:

$$\text{Adv}_{\mathcal{A}_{\text{ER}}, \text{RF}_i}^{\text{LEAK}}(\lambda) = \left| \Pr[\text{LEAK}(\Pi, i, \{P_1, \dots, P_n\}, \text{RF}_i, \lambda) = 1] - \frac{1}{2} \right|.$$

⁷ Looking ahead, in all our constructions the function Transform will typically be a very simple function like addition or field multiplication.

⁸ Note that, if we were to give P_i 's internal state when it gets adaptively corrupt instead of the composed state, the adversary can trivially distinguish since the party's state does not explain the sanitized transcript.

Fig. 6. Exfiltration-resistance game LEAK for a party P_i for adaptively secure MPCs



As in prior works on RF, we consider the notion of functionality-maintaining tampering attacks. Informally, such an attack excludes all conspicuous tamperings, which would otherwise be detected by honest parties. We provide a formal definition below (Def. 22).

We define *functionality-maintaining tampering* below.

Definition 22. Let Π be a multi-party protocol run between the parties P_1, \dots, P_n implementing functionality \mathcal{F} . We say that \overline{P}_i is a functionality maintaining tampering if, for all inputs $\mathbf{x} = (x_1, \dots, x_n)$, the output of execution $\Pi_{\overline{P}_i}(\mathbf{x})$ (where party P_i is replaced by tampering \overline{P}_i and all other parties are honest) is equal to $\mathcal{F}(\mathbf{x})$.

We also define *transparency* of reverse firewalls (Def.23) which was informally introduced in [DMS16], which means that the behavior of $\text{RF} \circ P$ is identical to the behavior of P if P is the honest implementation.

Definition 23. (Transparent RF in presence of adaptive corruptions). Let Π be a multi-party protocol run between the parties P_1, \dots, P_n satisfying functionality \mathcal{F} and having reverse firewalls RF_i for the set of honest parties $\{P_i\}_{i \in H}$. When the adversary (adaptively) corrupts a party P_i , it receives $\text{View}_{\text{RF}_i \circ P_i}$ as the view of P_i . Then $\forall i \in H$, we say that the firewall RF_i is transparent for party P_i against all other parties $\{P_j\}_{j \in [n] \setminus i}$, if for any PPT adversary A_{tr} , the advantage $\text{Adv}_{A_{\text{tr}}, \text{RF}_i}^{\text{TRANS}}(\lambda)$ of A_{tr} (defined below) in the game TRANS (see Fig. 7) is negligible in the security parameter λ .

The advantage of any adversary A_{tr} in the game TRANS is defined as:

$$\text{Adv}_{A_{\text{tr}}, \text{RF}_i}^{\text{TRANS}}(\lambda) = \left| \Pr[\text{TRANS}(\Pi, i, \{P_1, \dots, P_n\}, \text{RF}_i, \lambda, C) = 1] - \frac{1}{2} \right|.$$

We will also need the notion of *valid transcripts* and *detectable failures* of reverse firewalls, as presented in [DMS16]. We recall them below.

Definition 24 (Valid Transcripts [DMS16]). A sequence of bits r and private input I generate transcript \mathcal{T} in protocol Π if a run of the protocol Π 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 Π 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 .

Fig. 7. The Transparency game TRANS for a reverse firewall RF_i for party P_i for adaptively secure MPCs

TRANS($\Pi, i, \{P_1, \dots, P_n\}, \text{RF}_i, \lambda, \mathcal{C}$)

W.l.o.g, let $P_H = (P_1, \dots, P_h)$ denote the set of honest parties at the onset of the protocol Π , where $h = |H|$. The transparency game TRANS for a party $P_i \in P_H$ is modelled as an interactive game between a challenger \mathcal{C}_{tr} and an adversary \mathcal{A}_{tr} described as follows:

1. The adversary \mathcal{A}_{tr} provides the input I_H of all the honest parties to the challenger \mathcal{C}_{tr} .
2. The challenger \mathcal{C}_{tr} samples a bit $b \xleftarrow{\$} \{0, 1\}$ uniformly at random and does the following:
 - If $b = 1$, define $P_i^* \leftarrow \text{RF}_i \circ P_i$.
 - If $b = 0$, define $P_i^* \leftarrow P_i$.
3. \mathcal{C}_{tr} and \mathcal{A}_{tr} then engage in an execution of the MPC protocol Π , where the challenger \mathcal{C}_{tr} plays the role of all the honest parties P_H (with inputs I_H) and the adversary can adaptively corrupt parties in the set P_H . The \mathcal{C}_{tr} then returns the transcript \mathcal{T}^* of Π to \mathcal{A}_{ER} .
4. If \mathcal{A}_{tr} adaptively corrupts the party P_i at some point during the execution of Π (say at round k) the challenger \mathcal{C}_{tr} returns the composed view of P_i till round k , i.e., $\text{View}_{\text{RF}_i \circ P_i}^{\leq k}$ to \mathcal{A}_{ER} . Note that, $\text{View}_{\text{RF}_i \circ P_i}^{\leq k}$ can be efficiently constructed from $\text{View}_{P_i}^{\leq k}$ and the state st_{RF_i} of RF_i using the Transform function as a subroutine.
5. The game ends when \mathcal{A}_{tr} returns a bit b' as a guess for the bit b . Output 1 if $b' = b$.

Definition 25 (Detectable failure). A reverse firewall RF detects failure for party P in protocol Π if (a) $\Pi_{\text{RF} \circ P}$ has unambiguous transcripts; (b) the firewall outputs a special symbol \perp when run on any transcript that is not valid for $\Pi_{\text{RF} \circ P}$, and (c) there is a polynomial-time deterministic algorithm that decides whether a transcript \mathcal{T} is valid for $\Pi_{\text{RF} \circ P}$.

Input Replacement Tampering: We consider a special form of tampering attack, which we call the *input replacement tampering* attack, and observe that it is very difficult to construct any reverse firewall that preserves security (see Def. 20) for such class of tampering attacks in the context of protocols with *simulation-based* security requirements. Such tampering attacks work by substituting the actual input of the honest parties with a different (possibly (un)related) value. Let us illustrate this:

Let Π be a *two party* protocol run between parties P_0 and P_1 which securely computes some functionality f . Let P_1 be corrupt and the inputs of P_0 and P_1 be x and y respectively. Now, when P_0 is tampered to $\overline{P_0}$, its tampering observes x and replaces it with some x' before the execution of the protocol Π . Thus, after execution parties get $f(x', y)$ (instead of $f(x, y)$). To prove security preservation (SP) in this scenario, for any RF we will need to construct a simulator Sim that can generate the view of a corrupt P_1 interacting with $\text{RF} \circ \overline{P_0}$ in the real world. In the ideal world P_0 sends x as its input to the ideal functionality and hence, Sim receives $f(x, y)$. However, to generate the real world view Sim is required to generate the view corresponding to the output $f(x', y)$, without any knowledge of the tampering $\overline{P_0}$ or the input x . Note that, this example is not simply an artefact, but the problem arises since in the ideal world the simulator *does not* get access to the tampered implementations.

To get around this, in this work, we disallow this class of input replacement tampering attacks. However, we stress that this is *not* a limitation of our work as this assumption has been made implicitly in all the prior works on reverse firewalls which demand simulation-based security requirements [MS15, CDN20].

5 Relations between Security Preservation and Exfiltration Resistance (Static case)

In this section, we explore the relation between the notions of security preservation (SP) and exfiltration resistance (ER) for reverse firewalls in the MPC setting. Specifically, we show that

ER implies SP for MPC protocols for both static and adaptive corruptions; whereas the relation in the other direction is much less clear.

For all the implications we show in this section, we assume that security preservation be defined by the existence of a black-box simulator. We also assume that the adversaries are not computationally unbounded, and do not have access to additional oracles.

We first show the implication in the simpler case of static corruptions. Later, in Sec. 6 we prove the implication for adaptive case.

5.1 Exfiltration-Resistance implies Security Preservation

In this section, we show that exfiltration resistance implies security preservation for MPC protocols in the static corruption setting. We prove the following theorem.

Theorem 4. *Let f be an n -ary functionality and let Π be an n -party protocol that securely computes f with abort in presence of malicious adversaries. Let $C \subset [n]$ denote the indices of the corrupt parties in the protocol Π , and let $H = [n] \setminus C$ denote the indices of the honest parties at the outset of Π . Let $(\overline{P}_i)_{i \in H}$ denote the tampered implementations of the honest parties provided by the adversary. Also, let RF_i denote the RF corresponding to party P_i . Then for all $i \in H$, if RF_i is functionality maintaining, (strongly/weakly) exfiltration resistant for P_i against all other parties $\{P_j\}_{[n] \setminus i}$ and transparent (refer to Definitions 19, 16 and 18 respectively), then for all PPT adversaries \mathcal{A} and all PPT tamperings $(\overline{P}_i)_{i \in H}$ provided by \mathcal{A} , the firewalls RF_i for parties $\{P_i\}_{i \in H}$ (strongly/weakly) preserve security of the protocol Π according to Definition 17.*

Proof. For the proof of Theorem 4 we need to show that security of the MPC protocol Π is (strongly/weakly) preserved by the reverse firewalls RF_i for parties $\{P_i\}_{i \in H}$ by relying on (strong/weak) exfiltration-resistance of the firewalls RF_i , transparency of RF_i and the (static) security of the underlying MPC protocol Π . More formally, we will need to show that there exists a simulator/ideal-world adversary Sim such that for any real-world adversary \mathcal{A} participating in the protocol Π and maliciously corrupting a subset P_C of parties (where $C \subset [n]$ denotes the indices of the maliciously corrupted parties in Π), for all $\lambda \in \mathbb{N}$, inputs $\vec{x} \in (\{0, 1\}^\lambda)^n$ and auxiliary input $z \in \{0, 1\}^*$ the following two random variables are computationally indistinguishable:

$$\{\text{REAL}_{\Pi_{\{\text{RF}_i \circ \overline{P}_i\}_{i \in H}}, (\mathcal{C}, \mathcal{A})}(\lambda, \vec{x}, z)\} \approx_c \{\text{IDEAL}_{f, (\mathcal{C}, \text{Sim})}(\lambda, \vec{x}, z)\}, \quad (1)$$

We prove the above theorem via a sequence of hybrids, as described below. Let us denote this set of honest parties as $P_H = (P_{j_1}, \dots, P_{j_h})$, where $h = |H|$, and w.l.o.g., let us assume that the parties in P_H are ordered as above in some way, say in a lexicographic order.

- **Hyb₀** : This is the first hybrid which corresponds to the left hand side of Equation 1. In particular, **Hyb₀** corresponds to the real world view of the adversary \mathcal{A} in the MPC protocol Π , who corrupts the subset P_C of parties. All the honest parties in H are replaced with their corresponding tampered implementations composed with their firewalls, i.e., for all $i \in H$, P_i is replaced with $\text{RF}_i \circ \overline{P}_i$ in the protocol Π . So, the adversary \mathcal{A} obtains the following view:

$$\{\text{REAL}_{\Pi_{\{\text{RF}_i \circ \overline{P}_i\}_{i \in H}}, (\mathcal{C}, \mathcal{A})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0, 1\}^\lambda)^n} \quad (2)$$

- **Hyb₁** : **Hyb₁** is same as **Hyb₀**, except that, in the protocol Π the implementation of the first honest party P_{j_1} in the set P_H is replaced by its honest implementation composed with its

firewall RF_{j_1} . The rest of the honest parties remain tampered, that is, $\{P_{j_i}\}_{j_i \in \mathcal{H} \setminus \{j_1\}}$ are still replaced by $\text{RF}_{j_i} \circ \overline{P_{j_i}}$, as in Hyb_0 . In particular, the adversary \mathcal{A} obtains the following view:

$$\{\text{REAL}_{\Pi_{(\text{RF}_{j_1} \circ P_{j_1}, \{\text{RF}_{j_i} \circ \overline{P_{j_i}}\}_{j_i \in \mathcal{H} \setminus \{j_1\}})}, (\mathcal{C}, \mathcal{A})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0,1\}^\lambda)^n}$$

We now present the general description of the ℓ -th hybrid for all $1 \leq \ell \leq h$ as follows:

- Hyb_ℓ : In Hyb_ℓ , in the protocol Π the implementations of the first ℓ honest parties $\{P_{j_1}, P_{j_2}, \dots, P_{j_\ell}\}$ in the set $P_{\mathcal{H}}$ are replaced by their corresponding honest implementations composed with their firewalls. The rest of the honest parties $\{P_{j_{k'}}\}_{j_{k'} \in \mathcal{H} \setminus \{j_1, \dots, j_\ell\}}$ are still replaced by $\text{RF}_{j_{k'}} \circ \overline{P_{j_{k'}}}$ in the protocol Π , as in Hyb_0 . In particular, the adversary \mathcal{A} obtains the following view:

$$\{\text{REAL}_{\Pi_{(\{\text{RF}_{j_k} \circ P_{j_k}\}_{j_k \in \mathcal{H}, k \in [\ell]}, \{\text{RF}_{j_{k'}} \circ \overline{P_{j_{k'}}}\}_{j_{k'} \in \mathcal{H}, k' \in [\ell+1, h]}), (\mathcal{C}, \mathcal{A})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0,1\}^\lambda)^n} \quad (3)$$

Note that, when $\ell = 0$, we are in Hyb_0 , i.e., when the implementations of all the honest parties in $P_{\mathcal{H}}$ are replaced by their corresponding tampered implementations composed with their firewalls in the protocol Π . On the other hand, when $\ell = h$, we are in Hyb_h where the implementations of all the honest parties in $P_{\mathcal{H}}$ are replaced by their honest implementations composed with their firewalls. For the sake of completeness, we present the h -th hybrid as follows:

- Hyb_h : In Hyb_h , in the protocol Π the implementations of all the parties $\{P_{j_1}, \dots, P_{j_\ell}\}$ in the set $P_{\mathcal{H}}$ are replaced by their corresponding honest implementations composed with their firewalls. In particular, the adversary \mathcal{A} obtains the following view:

$$\{\text{REAL}_{\Pi_{(\{\text{RF}_{j_i} \circ P_{j_i}\}_{j_i \in \mathcal{H}}), (\mathcal{C}, \mathcal{A})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0,1\}^\lambda)^n}$$

Now, we prove the indistinguishability of consecutive hybrids.

Claim. $\forall 1 \leq \ell \leq h, \text{Hyb}_{\ell-1} \approx_c \text{Hyb}_\ell$

Proof. Note that, the two hybrids $\text{Hyb}_{\ell-1}$ and Hyb_ℓ differ in the implementation of the party $P_{j_\ell} \in P_{\mathcal{H}}$. In particular, in $\text{Hyb}_{\ell-1}$, the party P_{j_ℓ} is replaced with $\text{RF}_{j_\ell} \circ \overline{P_{j_\ell}}$ in Π ; whereas in Hyb_ℓ the party P_{j_ℓ} is replaced with $\text{RF}_{j_\ell} \circ P_{j_\ell}$. Let \mathcal{D}_ℓ be an adversary that distinguishes between these two hybrids. Using \mathcal{D}_ℓ , we construct an exfiltration resistant adversary \mathcal{A}_{ER} such that if the advantage of \mathcal{D}_ℓ is non-negligible, then the advantage of \mathcal{A}_{ER} in breaking the exfiltration-resistance game (Definition 16) is also non-negligible. The reduction is as follows:

- The adversary \mathcal{A}_{ER} receives the tampered implementations $\{\overline{P_{j_\ell}}, \overline{P_{j_{\ell+1}}}, \dots, \overline{P_h}\}$ corresponding to the last $(h - \ell + 1)$ honest parties in the set $P_{\mathcal{H}}$ from the distinguisher \mathcal{D}_ℓ .
- \mathcal{A}_{ER} then sets (1) $\overline{P_{j_k}} = \text{RF}_{j_k} \circ P_{j_k}$ for all $j_k \in \mathcal{H}$ and $k \in [\ell - 1]$, and (2) randomly samples inputs for the honest parties. He forwards the input set, and the set $\{\overline{P_{j_1}}, \dots, \overline{P_{j_{\ell-1}}}, \overline{P_{j_\ell}}, \dots, \overline{P_{j_h}}, \{P_C\}\}$ to the challenger \mathcal{C}_{ER} of the exfiltration-resistance (ER) game (see the LEAK game in Figure 4). In other words, \mathcal{A}_{ER} sets the tampered implementations of the first $\ell - 1$ honest parties in the set $P_{\mathcal{H}}$ to be simply their corresponding honest implementations and sets the implementations of the remaining $(h - \ell + 1)$ honest parties as received from \mathcal{D}_ℓ .
- Upon receiving the transcript \mathcal{T}^* and the states $\{\text{st}_{P_i}\}$ of all parties $\{P_{j_1}, \dots, P_{j_{\ell-1}}, P_{j_{\ell+1}}, \dots, P_{j_h}, \{P_C\}\}$ other than the party P_{j_ℓ} , \mathcal{A}_{ER} constructs the view as in Equation 3 (note that, \mathcal{A}_{ER} can efficiently construct this view given the states of all parties other than P_{j_ℓ} ⁹). It then forwards this view to the distinguisher \mathcal{D}_ℓ .

⁹ Note that, it only needs to know the states corresponding to the corrupt parties to construct the view as in Equation 3. Also, we know that the party P_{j_ℓ} is not corrupt, hence it is always possible to reconstruct this view.

- If \mathcal{D}_ℓ outputs a bit b' , the adversary \mathcal{A}_{ER} outputs the same bit b' . Note that, if the challenger \mathcal{C}_{ER} of the ER game sampled the bit $b = 1$, then we are in $\text{Hyb}_{\ell-1}$; whereas if $b = 0$, we are in Hyb_ℓ . Hence, if the advantage of \mathcal{D}_ℓ in distinguishing between these hybrids is non-negligible, the advantage of \mathcal{A}_{ER} in breaking the ER game is also non-negligible. \square

Note that, at the end of Hyb_h , all the honest parties $\{P_{j_i}\}_{i \in \text{H}}$ in the set P_{H} are replaced by $\text{RF}_{j_i} \circ P_{j_i}$.

- Hyb_{h+1} : Hyb_{h+1} is same as Hyb_h , except that, in the protocol Π all the honest parties $\text{H} = (P_{j_1}, \dots, P_{j_h})$ have honest implementations and there is no RF for the honest parties. In particular, the adversary \mathcal{A} obtains the following view: $\{\text{REAL}_{\Pi(\{P_{j_i}\}_{j_i \in \text{H}}), (\mathcal{C}, \mathcal{A})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0,1\}^\lambda)^n}$.

Claim. $\text{Hyb}_h \approx_c \text{Hyb}_{h+1}$.

Proof: It is easy to see that the hybrids Hyb_h and Hyb_{h+1} are identically distributed by relying on the transparency game of the firewalls $\{\text{RF}_{j_i}\}_{i \in \text{H}}$ (see the TRANS game in Figure 18). More formally, we can define a set of h hybrids and show that the consecutive hybrids are indistinguishable by the transparency properties of each of the reverse firewalls. \square

- Hyb_{h+2} : This is the final hybrid. This hybrid corresponds to the the ideal world adversary view for the MPC protocol Π where the set of corrupted parties is $\{P_i\}_{i \in \text{C}}$. All the honest parties $P_{\text{H}} = (P_{j_1}, \dots, P_{j_h})$ have honest implementations and there is no RF for the honest parties. In particular, the adversary \mathcal{A} obtains the following view:

$$\{\text{IDEAL}_{f, (\mathcal{C}, \text{Sim})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0,1\}^\lambda)^n} \quad (4)$$

Claim. $\text{Hyb}_{h+1} \approx_c \text{Hyb}_{h+2}$

Proof: Hyb_{h+2} is indistinguishable from Hyb_{h+1} due to the security of the protocol Π . Hyb_{h+1} corresponds to the real world adversary view of Π (without any RF) and Hyb_{h+2} corresponds to the ideal world adversary view of Π (without any RF). \square

Thus, combining Equations 2–4, we obtain Equation 1. This completes the proof of Theorem 4. \square

6 Relations between Security Preservation and Exfiltration Resistance (adaptive case)

In this section, we show that ER implies SP for MPC protocols for adaptive corruptions. Here too, we assume that security preservation be defined by the existence of a black-box simulator, the adversaries are not computationally unbounded, and do not have access to additional oracles. Looking ahead, all of our constructions will satisfy the above requirements.

6.1 Exfiltration-Resistance implies Security Preservation

In this section we show that exfiltration resistance implies security preservation for adaptively-secure MPC protocols by proving the following theorem.

Theorem 5. *Let f be an n -ary functionality and let Π be an n -party protocol that securely computes f with abort in presence of malicious adaptive adversaries. Let $\text{C} \subset [n]$ denote the indices of adaptively corrupt the corrupted parties in the protocol Π , and let $\text{H} = [n] \setminus \text{C}$ denote the indices of the honest parties at the outset of Π . Let $(\bar{P}_i)_{i \in \text{H}}$ denote the tampered implementations of the honest parties provided by the adversary. Also, let RF_i denote the RF corresponding to party P_i . Then for all $i \in \text{H}$, if RF_i is*

functionality maintaining, (strongly/weakly) exfiltration resistant with adaptive security for P_i against all other parties $\{P_j\}_{[n]\setminus i}$ and transparent (refer to Def. 19, 21 and 23 respectively), then for all PPT adversaries \mathcal{A} and all PPT tamperings $(\overline{P}_i)_{i \in H}$ provided by \mathcal{A} , the firewalls RF_i for parties $\{P_i\}_{i \in H}$ (strongly/weakly) preserve security of the protocol Π according to Def. 20 in the presence of adaptive corruptions.

Proof. We need to show that security of the MPC protocol Π is (strongly/weakly) preserved by the reverse firewalls RF_i for parties $\{P_i\}_{i \in H}$ by relying on (strong/ weak) exfiltration-resistance of the firewalls RF_i , transparency of RF_i and the adaptive security of the underlying MPC protocol Π . More formally, we will need to show that there exists a simulator/ideal-world adversary Sim such that for any real-world adversary \mathcal{A} participating in the protocol Π , adaptively (maliciously) corrupting parties during the execution, for all $\lambda \in \mathbb{N}$, inputs $\vec{x} \in (\{0, 1\}^\lambda)^n$ and auxiliary input $z \in \{0, 1\}^*$ the following two random variables are computationally indistinguishable:

$$\{\text{REAL}_{\Pi_{\{\text{RF}_i \circ \overline{P}_i\}_{i \in H}}, (\mathcal{C}, \mathcal{A})}(\lambda, \vec{x}, z)\} \approx_c \{\text{IDEAL}_{f, (\mathcal{C}, \text{Sim})}(\lambda, \vec{x}, z)\}, \quad (5)$$

Note that, in the above \mathcal{C} denotes the set of parties adaptively corrupted by the adversary. This set is allowed to grow during the execution of the protocol. H denotes the indices of honest parties at the outset of the protocol Π , i.e the initial set of honest parties.

We prove the above theorem via a sequence of hybrids, as described below.

- **Hyb₀** : This is the first hybrid which corresponds to the left hand side of Eq. 5. In particular, **Hyb₀** corresponds to the real world view of the adversary \mathcal{A} in the MPC protocol Π , who adaptively corrupts the subset P_C of parties. When the adversary \mathcal{A} corrupts some party $P_i \in P_H$, return $\text{View}_{\text{RF}_i \circ P_i} = \text{Transform}(\text{View}_{P_i}, \text{st}_{\text{RF}_i})$ to \mathcal{A} , and move P_i to the corrupt set. All the honest parties in H are replaced with their corresponding tampered implementations composed with their firewalls, i.e., for all $i \in H$, P_i is replaced with $\text{RF}_i \circ \overline{P}_i$ in the protocol Π . So, the view of the real world adversary \mathcal{A} consists of the following:

$$\{\text{REAL}_{\Pi_{\{\text{RF}_i \circ \overline{P}_i\}_{i \in H}}, (\mathcal{C}, \mathcal{A})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0, 1\}^\lambda)^n} \quad (6)$$

- **Hyb₁** : **Hyb₁** is same as **Hyb₀**, except that, in the protocol Π the implementation of the first party P_1 is replaced by its honest implementation composed with its firewall RF_1 . The rest of the honest parties remain tampered, that is, $\{P_j\}_{j \in H \wedge j \in \{2, \dots, n\}}$ are still replaced by $\text{RF}_j \circ \overline{P}_j$, and the corrupt parties remain as in **Hyb₀**. In particular, the view of the real world adversary is as follows:

$$\{\text{REAL}_{\Pi_{(\text{RF}_1 \circ P_1, \{\text{RF}_j \circ \overline{P}_j\}_{j \in H \wedge j \in \{2, \dots, n\}})}, (\mathcal{C}, \mathcal{A})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0, 1\}^\lambda)^n}$$

We now present the general description of the ℓ -th hybrid for all $1 \leq \ell \leq n$ as follows:

- **Hyb_ℓ** : In **Hyb_ℓ**, in the protocol Π the implementations of the first ℓ parties $\{P_1, P_2, \dots, P_\ell\}$ are replaced by their corresponding honest implementations composed with their firewalls. The other honest parties $\{P_j\}_{j \in H \wedge j \in \{\ell+1, \dots, n\}}$ are still replaced by $\text{RF}_j \circ \overline{P}_j$ in the protocol Π , as in **Hyb₀**. When the adversary \mathcal{A} corrupts a currently honest party P_j , return $\text{View}_{\text{RF}_j \circ P_j} = \text{Transform}(\text{View}_{P_j}, \text{st}_{\text{RF}_j})$ to \mathcal{A} , and move P_j to the set of corrupt parties as before. In particular, the adversary \mathcal{A} obtains the following view:

$$\{\text{REAL}_{\Pi_{(\{\text{RF}_j \circ P_j\}_{j \in [\ell]}, \{\text{RF}_j \circ \overline{P}_j\}_{j \in H \wedge j \in \{\ell+1, \dots, n\}})}, (\mathcal{C}, \mathcal{A})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0, 1\}^\lambda)^n} \quad (7)$$

Note that, when $\ell = 0$, we are in **Hyb₀**, i.e., when the implementations of all the honest parties in P_H are replaced by their corresponding tampered implementations composed with their firewalls in the protocol Π . On the other hand, when $\ell = n$, we are in **Hyb_n** where the implementations of all the honest parties are replaced by their honest implementations

composed with their firewalls. For the sake of completeness, we present the n -th hybrid as follows:

- **Hyb _{n}** : In **Hyb _{n}** , in the protocol Π the implementations of all the honest parties $\{P_j\}_{j \in H}$ are replaced by their corresponding honest implementations composed with their firewalls. In particular, the adversary \mathcal{A} obtains the following view:

$$\{\text{REAL}_{\Pi(\{\text{RF}_j \circ P_j\}_{j \in H}), (\mathcal{C}, \mathcal{A})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0,1\}^\lambda)^n} \quad (8)$$

Note that, in each subsequent hybrid we replace each party (honest and corrupt) with the fire-walled honest implementation. However this does not mean that the corrupt parties are forced to behave with honest implementation. As will be clear in the indistinguishability proof below, this approach does not enforce any restriction on the way the corrupt parties behave. We take this approach for readability and ease of proving indistinguishability.

Now, we prove the indistinguishability of consecutive hybrids.

Claim. $\forall 1 \leq \ell \leq n, \text{Hyb}_{\ell-1} \approx_c \text{Hyb}_\ell$

Proof. Note that, the two hybrids **Hyb _{$\ell-1$}** and **Hyb _{ℓ}** differ in the implementation of the party P_ℓ . In particular, the only change from **Hyb _{$\ell-1$}** to **Hyb _{ℓ}** is that $\text{RF}_\ell \circ \overline{P}_\ell$ in the former is replaced by $\text{RF}_\ell \circ P_\ell$ in the latter. Let \mathcal{D}_ℓ be an adversary that distinguishes between these two hybrids. Since the adversary is allowed to corrupt parties adaptively, assume that party P_ℓ is corrupted by \mathcal{D}_ℓ in round k_ℓ . Using \mathcal{D}_ℓ , we construct an exfiltration resistant adversary \mathcal{A}_{ER} such that if the advantage of \mathcal{D}_ℓ is non-negligible, then the advantage of \mathcal{A}_{ER} in breaking the exfiltration-resistance game (Def. 21) is also non-negligible. At a high level, \mathcal{A}_{ER} interacts with \mathcal{D}_ℓ as the challenger for the indistinguishability game for \mathcal{D}_ℓ so that ultimately \mathcal{D}_ℓ either sees views from **Hyb _{$\ell-1$}** or **Hyb _{ℓ}** . If party P_ℓ is already in the corrupt set at the start of the protocol, then exfiltration-resistance is trivially satisfied for P_ℓ . Otherwise, for the case when P_ℓ gets adaptively corrupt in round k_ℓ , by the exfiltration guarantee, \mathcal{D}_ℓ cannot distinguish views till round k_ℓ .

The reduction is as follows:

- The adversary \mathcal{A}_{ER} receives the initial indices of honest parties (H), and the tampered implementations $\{\overline{P}_j\}_{j \in H \wedge j \in \{\ell, \dots, n\}}$ corresponding to the last $(|H| - \ell + 1)$ honest parties in the set P_H from the distinguisher \mathcal{D}_ℓ .
- \mathcal{A}_{ER} then sets (1) $\overline{P}_j = \text{RF}_j \circ P_j$ for all $j \in H \wedge j \in [\ell - 1]$, and (2) randomly samples inputs for the parties in the honest set to define I and sends it to \mathcal{C}_{ER} . It forwards the set $\{\overline{P}_j\}_{j \in H}$ (here H is set of honest parties at the outset of Π) to the challenger \mathcal{C}_{ER} of the exfiltration-resistance (ER) game (see the LEAK game in Fig. 6). In other words, \mathcal{A}_{ER} sets the tampered implementations of the first $\ell - 1$ honest parties in the set P_H to be simply their corresponding honest implementations with a wrapper of firewall on top of it and sets the implementations of the remaining $(|H| - \ell + 1)$ honest parties as received from \mathcal{D}_ℓ .
- Now \mathcal{A}_{ER} interacts with the challenger \mathcal{C}_{ER} and the distinguisher \mathcal{D}_ℓ to execute Π . \mathcal{C}_{ER} executes Π on the behalf of the currently honest parties, and \mathcal{D}_ℓ executes on behalf of the currently dishonest parties. \mathcal{A}_{ER} passes round messages between \mathcal{C}_{ER} and \mathcal{D}_ℓ . If \mathcal{D}_ℓ adaptively corrupts an honest party, \mathcal{A}_{ER} too corrupts the same party and receives a transformed view from \mathcal{C}_{ER} which it passes on to \mathcal{D}_ℓ . On corruption of an honest party, \mathcal{C}_{ER} moves it from the set of honest to dishonest parties. Note that if party P_ℓ is statically corrupt, the indistinguishability in the ER game trivially follows.

- Upon receiving the final views of parties from \mathcal{C}_{ER} as described in the LEAK game in Fig. 6, \mathcal{A}_{ER} constructs the view as in Eq. 7. If \mathcal{D}_ℓ corrupts any party post execution, \mathcal{A}_{ER} forwards relevant view. Note that, \mathcal{A}_{ER} only needs to know the views corresponding to the corrupt parties to construct the view as in Eq. 7.
- If \mathcal{D}_ℓ outputs a bit b' , the adversary \mathcal{A}_{ER} outputs the same bit b' . Note that, if the challenger \mathcal{C}_{ER} of the ER game sampled the bit $b = 1$, then we are in $\text{Hyb}_{\ell-1}$; whereas if $b = 0$, we are in Hyb_ℓ . Hence, if the advantage of \mathcal{D}_ℓ in distinguishing between these hybrids is non-negligible, the advantage of \mathcal{A}_{ER} in breaking the ER game is also non-negligible. \square

Note that, at the end of Hyb_n , all the honest parties $\{P_j\}_{j \in \text{H}}$ in the set P_{H} are replaced by $\text{RF}_j \circ P_j$.

- Hyb_{n+1} : Hyb_{n+1} is same as Hyb_n , except that, in the protocol Π all the honest parties in H have honest implementations and there is no RF for the honest parties. In particular, the adversary \mathcal{A} obtains the following view:

$$\{\text{REAL}_{\Pi(\{P_j\}_{j \in \text{H}}), (\mathcal{C}, \mathcal{A})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0,1\}^\lambda)^n}.$$

Claim. $\text{Hyb}_n \approx_c \text{Hyb}_{n+1}$.

Proof: It is easy to see that the hybrids Hyb_n and Hyb_{n+1} are identically distributed by relying on the transparency of the firewalls $\{\text{RF}_j\}_{j \in \text{H}}$ (see Def. 23 for the formal definition). More formally, we can define a set of n hybrids (similar to the claim earlier in this section) and show that the consecutive hybrids are indistinguishable by the transparency property of each of the reverse firewalls. \square

- Hyb_{n+2} : This is the final hybrid. This hybrid corresponds to the the ideal world adversary view for the MPC protocol Π where the set of corrupted parties is $\{P_i\}_{i \in \text{C}}$. All the honest parties P_{H} have honest implementations and there is no RF for the honest parties. In particular, the adversary \mathcal{A} obtains the following view:

$$\{\text{IDEAL}_{f, (\mathcal{C}, \text{Sim})}(\vec{x})\}_{\lambda \in \mathbb{N}, \vec{x} \in (\{0,1\}^\lambda)^n} \quad (9)$$

Claim. $\text{Hyb}_{n+1} \approx_c \text{Hyb}_{n+2}$

Proof: Hyb_{n+2} is indistinguishable from Hyb_{n+1} due to the security of the protocol Π . Hyb_{n+1} corresponds to the real world adversary view of Π (without any RF) and Hyb_{n+2} corresponds to the ideal world adversary view of Π (without any RF). \square

Thus, combining the above three claims, we obtain Eq. 5. This completes the proof of Thm.

5. \square

Limitation in proving SP implies ER. Can we hope to show equivalence of the two notions? Considering the current definitions of ER and SP this seems infeasible. The reason is that the ER adversary can choose the inputs of the tampered honest parties I_{H} , whereas the SP adversary only controls the corrupt party's input set. It is difficult to construct an SP adversary given an ER adversary because the SP adversary cannot simulate an ER challenge session corresponding to honest input set I_{H} (returned by ER adversary) given an SP challenge session. Moreover, another issue is that the ER adversary expects to receive the views of all the parties, except the challenged party P_i (unless P_i gets corrupt). The SP adversary, however, receives the views of the corrupt parties, and the reduction will fail to simulate the views of the honest parties to the ER adversary.

7 Adaptively-Secure Compiler using Reverse Firewalls in the urs model

In this section, we show a compiler that transforms any semi-honest adaptively secure MPC protocol to a maliciously-secure MPC protocol in the urs model, which withstands adaptive corruptions and admits reverse firewalls. As a building block, we first present the adaptively-secure multiparty augmented coin tossing protocol using reverse firewalls.

7.1 Adaptively-Secure Augmented Coin-Tossing using Reverse Firewalls in the urs model

The adaptively-secure augmented coin tossing protocol $\Pi_{\text{a-coin}}$ is used to generate *random bits* (according to Def. 14) for all the parties participating in the adaptively-secure MPC protocol. The initiating party receives a random tuple (S, R) and all other parties receive a commitment $\text{Com}(S; R)$ (under the urs of party P_i) of S using commitment randomness R , where $S \in \{0, 1\}^\lambda$, Com is an adaptively secure homomorphic commitment and $R \leftarrow \mathcal{R}_{\text{Com}}$. The protocol $\Pi_{\text{a-coin}}$ is presented in Fig. 8.

Fig. 8. Adaptively-Secure Multi-party Augmented Coin-Tossing Protocol $\Pi_{\text{a-coin}}$ for P_i

<ul style="list-style-type: none"> – Primitives: $(\text{Gen}, \text{Com}, \text{Verify})$ is an adaptively secure homomorphic commitment scheme where $\text{Com}(\text{urs}, a_1; b_1) \cdot \text{Com}(\text{urs}, a_2; b_2) = \text{Com}(\text{urs}, a_1 + a_2; b_1 + b_2)$ for $a_1, a_2 \in \{0, 1\}^\lambda$ and $b_1, b_2 \in \mathcal{R}_{\text{Com}}$ respectively. We denote $\text{Com}_i(m; r) = \text{Com}(\text{urs}_i, m; r)$ under $\text{urs}_i \leftarrow \text{Gen}(1^\lambda)$. – Public Inputs: Each party gets as input $\text{urs}_{\text{a-coin}} = \{\text{urs}_i\}_{i \in [n]}$ where $\text{urs}_i \leftarrow \text{Com.Gen}(i, 1^\lambda)$. Party P_i commits using Com_i in the protocol.
<p>Round 1: For $j \in [n] \setminus i$, every party P_j samples $s_j \leftarrow \{0, 1\}^\lambda$ and $r_j \leftarrow \mathcal{R}_{\text{Com}}$ respectively. It computes $c_j = \text{Com}_j(s_j; r_j)$, and broadcasts c_j.</p> <p>Round 2: Party P_i chooses a random $s_i \leftarrow \{0, 1\}^\lambda$. It then computes $c_i = \text{Com}_i(s_i; r_i)$ for a random $r_i \leftarrow \mathcal{R}_{\text{Com}}$, and broadcasts c_i.</p> <p>Round 3: For $j \in [n] \setminus i$, every party P_j broadcasts (s_j, r_j) as the opening of c_j.</p> <p>Local Computation:</p> <ul style="list-style-type: none"> – For $j \in [n]$, P_j aborts if $\exists k \in [n] \setminus i$ s.t. $\text{Verify}(\text{urs}_k, c_k, s_k, r_k) = \perp$. – Party P_i sets $S = \sum_{i \in [n]} s_i$ and $R = \sum_{i \in [n]} r_i$. P_i outputs $C = \text{Com}_i(S; R)$. – For $j \in [n] \setminus i$, Party P_j sets $S_j = \sum_{k \in [n] \setminus i} s_k$ and $R_j = \sum_{k \in [n] \setminus i} r_k$. P_j outputs $C = c_i \cdot \text{Com}_i(S_j; R_j)$.

Theorem 6. *Assuming Com is an adaptively secure homomorphic commitment in the urs model, $\Pi_{\text{a-coin}}$ securely implements the augmented coin-tossing functionality (Def. 14) against adaptive corruption of parties in the urs model.*

Proof. We consider a set H of honest parties and a set C of statically corrupt parties. We have two exhaustive cases of corruptions:

P_i^* is statically corrupt. In this case, $P_i \in C$ is statically corrupt and there exists a set of honest parties $\{P_\ell\}_{\ell \in H}$. The simulator Sim receives a random string S from the functionality and it needs to construct a view s.t. it outputs S . Sim constructs simulated commitments $\{c_\ell\}_{\ell \in H}$ on behalf of the honest parties in the first round by invoking the simulator of the commitment functionality. It receives the commitments $\{c_k\}_{k \in C}$ of the adversarial parties at the end of first and second rounds. It extracts the underlying $\{s_k\}_{k \in C}$ values by invoking the simulator of the commitment functionality. In the third round the simulator Sim equivocates the $\{c_\ell\}_{\ell \in H}$ commitments by invoking the Com simulator s.t. the homomorphic addition of all the commitments equals S . When a party P_ℓ gets adaptively corrupted Sim opens the existing view of P_ℓ . The formal simulator can be found in 9 and the hybrids follow:

- Hyb_0 : This is the real world execution of the coin-tossing protocol.
- Hyb_1 : Same as Hyb_0 , except the simulator aborts if it fails to extract the messages from $\{c_k\}_{k \in C}$.

Indistinguishability. The simulator aborts if the simulator of Com_k fails to extract the adversarially committed messages in Hyb_1 . Thus, indistinguishability follows between the two hybrids due to the security of Com_k against a statically corrupt committer.

- Hyb_2 : Same as Hyb_1 , except the simulator constructs simulated commitments $\{c_\ell\}_{\ell \in H}$ by invoking the Com_ℓ simulator and equivocates following the simulation algorithm. It aborts if equivocation fails for any simulated commitment. Adaptive corruption is handled following the simulation algorithm. This is the ideal world view of the execution.

Indistinguishability. The simulator aborts if the simulator of Com_ℓ fails to equivocate the simulated commitments or if an adaptive adversary can distinguish between a simulated commitment (equivocated to a commitment on a random message) and an honestly generated commitment on a random message. Thus, indistinguishability follows between the two hybrids due to adaptive security of Com_ℓ .

Fig. 9. Simulation against a statically corrupt P_i^*

<p>Sim receives S as the output of the coin tossing-functionality and he is required to bias the output of the protocol to output S.</p> <hr/> <p><i>Round 1:</i> Sim simulates on behalf of the honest parties $\{P_\ell\}_{\ell \in H}$ as follows.</p> <ul style="list-style-type: none"> – Invokes the simulator of Com_ℓ to obtain simulated commitments $\{c_\ell\}_{\ell \in H}$ and their openings $\{s'_\ell, r'_\ell\}_{\ell \in H}$. – Broadcasts $\{c_\ell\}_{\ell \in H}$ to other parties. <p><i>Round 2:</i> Sim receives c_i from P_i^*.</p> <p><i>Round 3:</i> Sim has received $\{c_k\}_{k \in C}$ from the corrupt parties $\{P_k\}_{k \in C}$. It performs the following:</p> <ul style="list-style-type: none"> – It extracts $\{s'_k\}_{k \in C}$ from $\{c_k\}_{k \in C}$ by invoking the simulator of Com_k on $\{c_k\}_{k \in C}$. It aborts if extraction fails for any c_k. – Sim sets $\bar{S} = S - \sum_{k \in C} s'_k$. – Sim samples $\{s_\ell\}_{\ell \in H}$ randomly s.t. $\bar{S} = \sum_{\ell \in H} s_\ell$. – Sim invokes the simulator of Com_ℓ on $\{c_\ell\}_{\ell \in H}$ with input $\{s'_\ell, r'_\ell, s_\ell\}_{\ell \in H}$ to obtain $\{r_\ell\}_{\ell \in H}$. Sim aborts if $r_\ell = \perp$ for any $\ell \in C$. – For $\ell \in H$, Sim broadcasts (s_ℓ, r_ℓ) to other parties on behalf of P_ℓ. <p><i>Local Computation:</i></p> <ul style="list-style-type: none"> – Sim aborts if $\exists k \in [n]$ s.t. $s_k \neq s'_k$. – Else, it outputs $C = c_i \cdot \text{Com}_i(\sum_{j \in [n] \setminus i} s_j; \sum_{j \in [n] \setminus i} r_j)$ following the honest protocol. <p><i>Adaptive Corruption:</i></p> <p>If a party $P_\ell \in C$ gets corrupted adaptively before c_ℓ is opened (in Round 3), Sim samples $s_\ell \leftarrow \{0, 1\}^\lambda$ and invokes the simulator of Com_ℓ on $\{c_\ell\}_{\ell \in H}$ with input $\{s'_\ell, r'_\ell, s_\ell\}_{\ell \in H}$ to obtain $\{r_\ell\}_{\ell \in H}$. Sim aborts if $r_\ell = \perp$. Else, Sim provides $\{s_\ell, r_\ell\}$ as the internal state of P_ℓ.</p>

P_i gets corrupted post-execution. In this case, Sim simulates the honest parties $\{P_\ell\}_{\ell \in H \setminus i}$ honestly. It constructs a simulated commitment c_i for P_i by invoking the Com_i simulator. Later, when P_i gets corrupted post-execution and Sim is required to open the transcript to S , Sim observes the $\{s_j\}_{j \in [n] - i}$ values and Sim equivocates c_i accordingly. The formal simulator has been provided in 10 and the hybrids follow:

- Hyb_0 : This is the real world execution of the coin-tossing protocol.

- Hyb_1 : Same as Hyb_0 , except the simulator constructs a simulated commitment only for P_i and later when post-execution corruption of P_i occurs, Sim equivocates the simulated commitment by invoking the simulator of Com_i . Sim aborts if equivocation fails. This is the ideal world execution.

Indistinguishability. The simulator aborts if the simulator of Com_i fails to equivocate a simulated commitment in the presence of an adaptive adversary. Thus, indistinguishability follows between the two hybrids due to adaptive security of Com_i .

Fig. 10. Simulation against post-execution corruption of P_i

<p>If a party $P_\ell \in \mathcal{H} - i$ gets corrupted adaptively Sim opens the existing view of P_ℓ to the adversary.</p> <hr/> <p><i>Round 1:</i> Sim simulates the honest parties $\{P_\ell\}_{\ell \in \mathcal{H} - i}$ honestly.</p> <p><i>Round 2:</i> Sim constructs a simulated commitment c_i and receives its openings (s'_i, r'_i) by invoking the Com_i simulator.</p> <p><i>Round 3:</i> Sim simulates $\{P_\ell\}_{\ell \in \mathcal{H} - i}$ honestly.</p> <p><i>Local Computation:</i> Sim outputs C following the honest protocol.</p> <p><i>Post-Execution Corruption of P_i:</i> Sim obtains S from the functionality. Sim computes $s_i = S - \sum_{j \in [n] - i} s_j$ and invokes the Com_i simulator with input (c, s'_i, r'_i, s_i) to obtain opening r_i s.t. $c = \text{Com}_i(s_i; r_i)$. Sim aborts if $r_i = \perp$. Sim forwards (s_i, r_i) as the view of P_i to the adversary.</p> <p><i>Adaptive Corruption of $\{P_\ell\}_{\ell \in \mathcal{H} \wedge \ell \neq i}$:</i> Sim opens honestly constructed view of P_ℓ.</p>

□

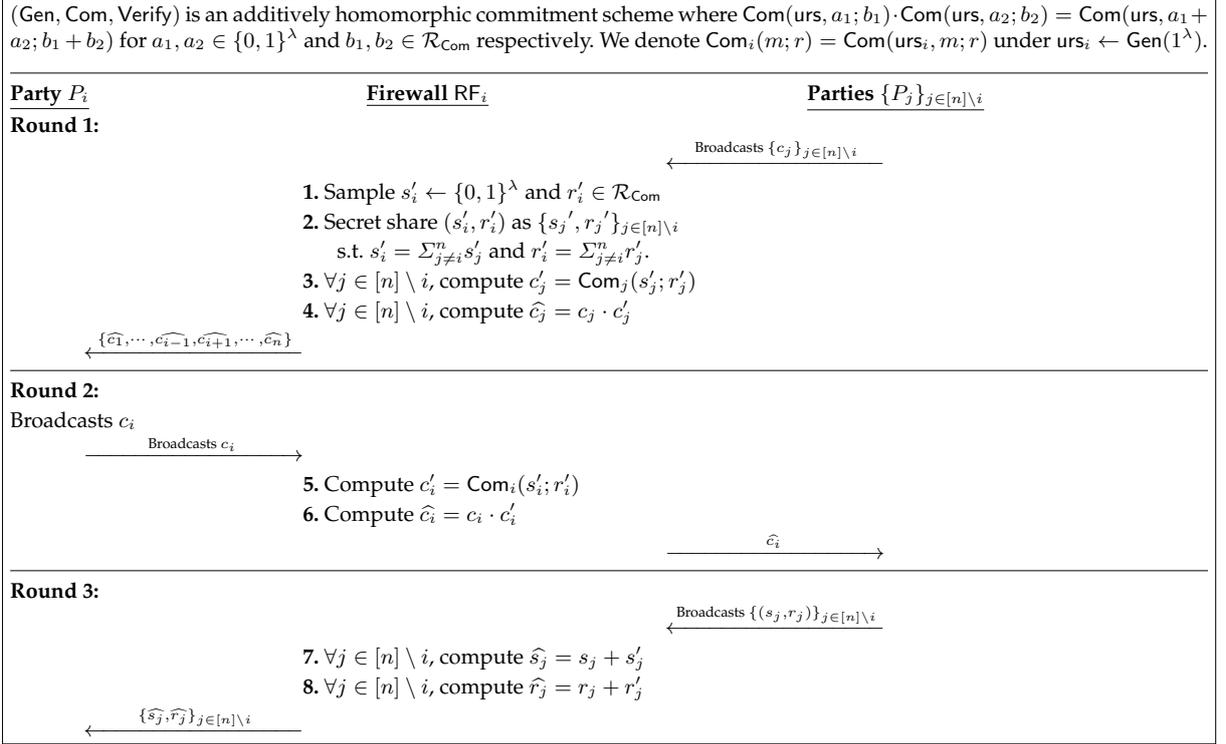
Next, we consider security of the protocol when honest parties are tampered. Our firewall RF_i for a tampered initiating party P_i and firewall $\{\text{RF}_k\}_{k \in [n] \setminus i}$ for a tampered receiving party $\{P_k\}_{k \in [n] \setminus i}$ is presented in Fig. 11 and Fig. 12 respectively. We prove that the firewalls provide weak exfiltration resistance and preserve security.

Theorem 7. *If the commitment scheme Com is adaptively secure in the urs model and is additively homomorphic, the firewall RF_i , (resp. RF_k) is transparent, functionality-maintaining, and provides weak exfiltration resistance for initiating P_i (resp. receiving party P_k) against other parties in $\Pi_{a\text{-coin}}$ with valid transcripts, and detects failure for P_i (resp. P_k).*

Proof. When the implementation of the initiating party P_i is honest he receives a random commitment C of a random message S under randomness R and the corresponding (S, R) . The other parties obtain C . If there is a RF attached to P_i then P_i receives a random commitment \widehat{C} of a random message \widehat{S} under randomness \widehat{R} and the corresponding $(\widehat{S}, \widehat{R})$. The other parties obtain \widehat{C} . When an initiating party P_i gets corrupted, the Transform algorithm takes P_i 's state (s_i, r_i) and the state of the firewall RF_i as (s'_i, r'_i) and returns $(\widehat{s}_i, \widehat{r}_i) = (s_i + s'_i, r_i + r'_i)$ as the state of the composed party. When a receiving party P_j gets corrupted, the Transform algorithm takes P_j 's state (s_j, r_j) and the state of the firewall RF_j as (s'_j, r'_j) and returns $(\widehat{s}_j, \widehat{r}_j) = (s_j + s'_j, r_j + r'_j)$ as the state of the composed party. This proves functionality maintaining property and the firewall is transparent too.

Weak exfiltration resistance can be argued based on the distribution of the views - $\text{RF}_i \circ P_i$ (resp. $\text{RF}_j \circ P_j$) and $\text{RF}_i \circ P'_i$ (resp. $\text{RF}_j \circ P'_j$) using the hiding property of the commitment scheme. Given a Commitment $C = \text{Com}(S; R)$ to a tampered message S under tampered randomness R , the composition of RFs (of honest parties) rerandomize C to obtain $\widehat{C} = C.\text{Com}(\widehat{S}; \widehat{R})$ where \widehat{S} and \widehat{R} are guaranteed to be random due to the homomorphic property of the commitment

Fig. 11. Reverse firewall RF_i for initiating party P_i involved in $\Pi_{\text{a-coin}}$ (from Fig. 8).



scheme. Thus, \widehat{C} is indistinguishable from an honestly generated random commitment to a random message. Our hybrid argument is summarized as follows.

$$\begin{aligned} \text{RF}_i \circ P_i &= (S, R, \text{Com}_i(S; R)) \equiv (U_\lambda, U_{\mathcal{R}_{\text{Com}}}, \text{Com}(U_\lambda; U_{\mathcal{R}_{\text{Com}}})) \\ &\approx (\widehat{S}, \widehat{R}, \text{Com}_i(\widehat{S}; \widehat{R})) = \text{RF}_i \circ P'_i \end{aligned}$$

Finally, we discuss security preservation for $\Pi_{\text{a-coin}}$ in the RF setting. We have shown in Thm. 6 that $\Pi_{\text{a-coin}}$ is adaptively secure and we know that it is weakly-exfiltration resistant. By applying the result of Thm. 5, it is proven that the RF preserves adaptive security of underlying $\Pi_{\text{a-coin}}$ protocol. \square

Now, consider $\Pi_{\text{a-coin}}^{\text{RF}}$, a firewalled version of the protocol where all honest parties P_i have their respective firewalls RF_i attached to them. Now, from Thm. 7, and our implication from Thm. 5, we conclude weak security preservation; we have that $\Pi_{\text{a-coin}}^{\text{RF}}$ securely implements augmented coin tossing in the presence of adaptive corruptions.

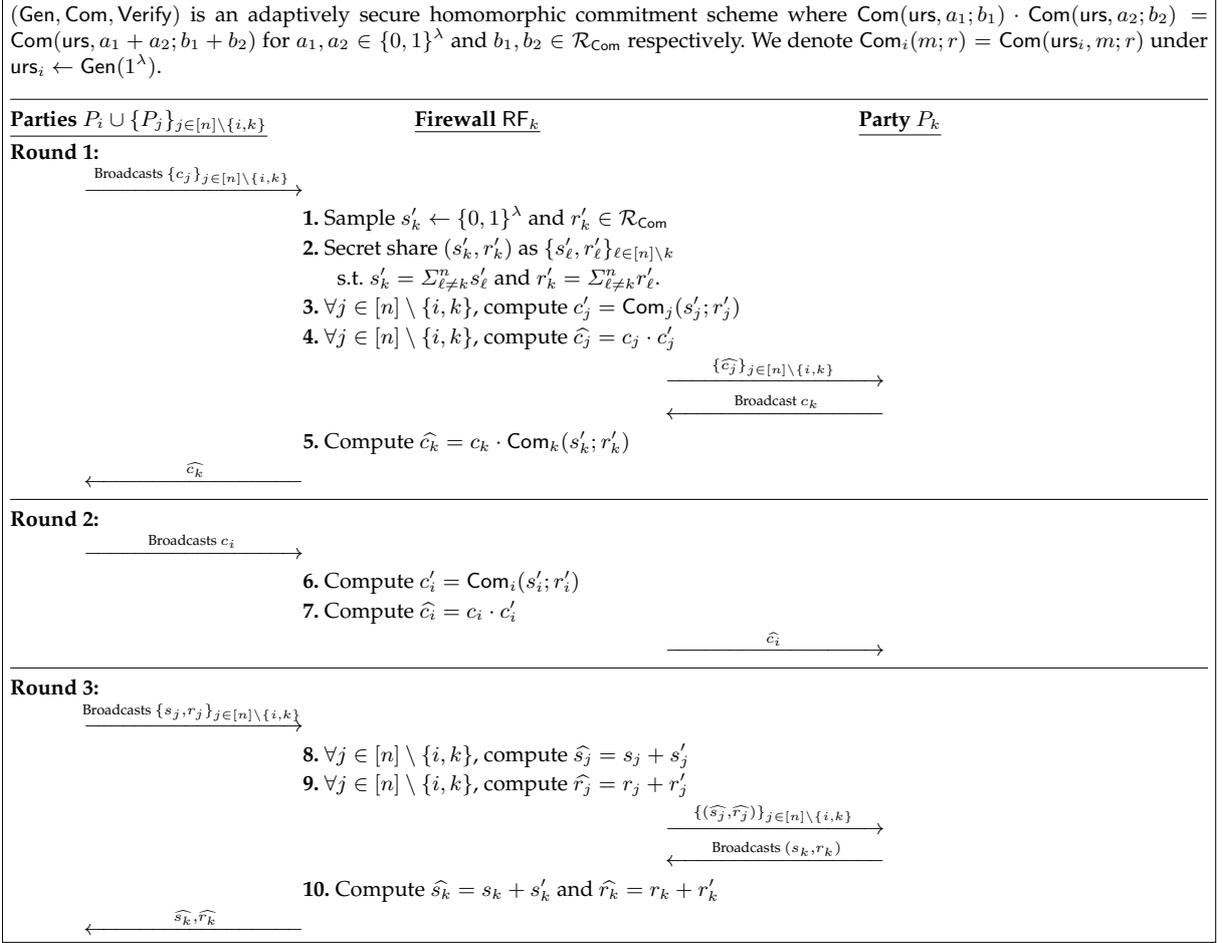
Theorem 8. $\Pi_{\text{a-coin}}^{\text{RF}}$ securely implements the augmented coin-tossing functionality (Def. 14) in the urs model against adaptive corruption of parties, and in the presence of functionality-maintaining tampering of honest parties.

Instantiation. We instantiate the adaptively secure homomorphic commitment in the urs model using the recent construction of [CSW20a]. It is additively homomorphic in the message and randomness space and can be instantiated based on DDH assumption in the urs model.

7.2 Adaptively-Secure ZK in the urs_{zk} model

We construct our adaptively secure ZK protocol Π_{zk} in the common random string model based on the recent ZK protocol of [CSW20b] by incorporating a coin tossing protocol to gen-

Fig. 12. Reverse firewall RF_k for receiving party P_k involved in $\Pi_{a\text{-coin}}$ (from Fig. 8).



erate the verifier's challenge. We first recall protocol of [CSW20b] below and then present our construction.

ZK Protocol of [CSW20b] The work of [CSW20b] construct an adaptively secure ZK protocol from equivocal commitments and PKE with oblivious ciphertext sampleability in the urs model. We briefly recall their protocol and then propose our modifications.

The Protocol. Let \mathcal{L}_{Ham} be the set of Hamiltonian graphs. The prover P possesses an n -node graph G . He interacts with the verifier V to prove that the graph is hamiltonian, i.e. $G \in \mathcal{L}_{\text{Ham}}$, given a Hamiltonian cycle ω as a witness. We describe one run of the protocol and it is repeated $\mathcal{O}(\lambda)$ times in parallel. The urs contains urs_{com} , the setup string of an equivocal commitment, and pk , public key of a PKE scheme. P samples a random n -node cycle H . He commits to the adjacency matrix of H by setting the commitments corresponding to edges as $\text{Com}(1; r)$ and others as $\text{Com}(0; r)$. The prover also encrypts the randomness for each commitment. For each commitment $c_j = \text{Com}(b_j; r_j)$ ($j \in [n^2]$) the prover encrypts the randomness as $E_{j, b_j} = \text{Enc}(\text{pk}, r_j; s_j)$ and samples E_{j, b_j} obliviously. P sends these commitments and encryptions as the first message a . V samples a random challenge bit e and sends it to the prover. Based on e , the prover performs the following:

- If $e = 0$: The prover decommits to the cycle H by opening the commitments corresponding to the edges in H and the encryption of the randomness corresponding to those commit-

ments. For each edge $j \in H$, prover sends $(1, r_j, s_j)$ and claims that $E_{j,0}$ was obviously sampled.

- If $e = 1$: The prover samples a random permutation γ of the witness cycle to H and opens to the non-edges in $\gamma(G)$ by opening the corresponding commitments and the encryptions (of commitment randomness). For each non-edge $j \in \overline{\gamma(G)}$, prover sends $(0, r_j, s_j)$ and claims that $E_{j,1}$ was obviously sampled. The prover also sends the permutation γ .

Upon obtaining $z = \{b_j, r_j, s_j\}_{j \in H}$ or $z = \{b_j, r_j, s_j\}_{j \in \overline{\gamma(G)}}$ values, V does the following:

- If $e = 0$: For all $j \in H$, verify that the prover has decommitted to the edges of a valid cycle by checking $c_j = \text{Com}(1; r_j)$ and $E_{j,1} = \text{Enc}(\text{pk}, r_j; s_j)$ and H is a valid n -node cycle.
- If $e = 1$: For all $j \in \overline{\gamma(G)}$, verify that the prover has decommitted to the non-edges of the permuted graph by verifying $c_j = \text{Com}(0; r_j)$ and $E_{j,0} = \text{Enc}(\text{pk}, r_j; s_j)$. V also verifies that decommitted edges correspond to non-edges in $\gamma(G)$.

Soundness of the protocol follows since the verifier challenge bits are random and the commitment scheme is computationally binding. The simulator can extract a valid witness from an accepting proof by decrypting the decommitments from the $(E_{j,0}, E_{j,1})$. Adaptive security follows from the equivocal property of Com and the oblivious sampleability of the encryption scheme. We refer to [CSW20b] for full details.

Our construction The ZK protocol of [CSW20b] is not secure when the implementation of the prover or the verifier is tampered. It admits a firewall which can rerandomize the prover's messages. But it cannot rerandomize the verifier's challenge bit, since the firewall will have to maul the response provided by the prover if the challenge bit is modified. The firewall cannot perform this task, since it does not know the witness. This problem arises since the original challenge bit is different from the challenge bit (rerandomized by the firewall) obtained by the prover. We modify the protocol, where the verifier's message is generated as the result of a coin-tossing protocol. The prover encrypts a random coin in the first message, verifier sends his random coin in the second message and prover opens to his encrypted coin in the third message. The final challenge coin is the xor of the two coins. The firewall rerandomizes the protocol steps of the coin-tossing s.t. the challenge is indeed random and the parties obtain the same rerandomized challenge. This allows the firewall to rerandomize the verifier's share of the challenge bit without mauling the prover's response. The above described protocol is repeated $\mathcal{O}(\lambda)$ times to obtain our final ZK protocol under reverse firewalls. The complete protocol can be found in Fig. 13, 14.

The security of our protocol is summarized below.

Theorem 9. *If Com is a non-interactive equivocal commitment scheme in the urs model and PKE is an IND-CPA public key encryption scheme (where the public key is statistically close to a random string) with oblivious ciphertext sampleability, then Π_{ZK} realizes \mathcal{F}_{ZK} for all NP relations against adaptive corruptions in the urs model.*

Proof Sketch. Our protocol is similar to the protocol of [CSW20b] except that we add a coin-tossing protocol to generate e . A corrupt prover cannot bias the coin e since the verifier's share of the coin is sampled randomly by the verifier after obtaining the prover's encrypted share of the coin. The prover cannot open its share to a different value since the encryption is perfectly binding. This guarantees security against a statically corrupt prover. When the verifier is corrupt, the simulator of [CSW20b] can be invoked to simulate the proof. The [CSW20b] ZK simulator relies on the equivocal property of Com and oblivious ciphertext sampleability of

Fig. 13. Adaptively Secure Rerandomizable ZK Protocol

Π_{zk}

- **Primitives:** Com = (Gen, Com, Verify) is an additively homomorphic equivocal commitment scheme where $\text{Com}(\text{urs}, a_1; b_1) \cdot \text{Com}(\text{urs}, a_2; b_2) = \text{Com}(\text{urs}, a_1 + a_2; b_1 + b_2)$ for $a_1, a_2 \in \{0, 1\}^\lambda$ and $b_1, b_2 \in \mathcal{R}_{\text{Com}}$ respectively. PKE = (Gen, Enc, Dec) is an additively homomorphic IND-CPA encryption with oblivious ciphertext sampleability where $\text{Enc}(\text{pk}, m_1; r_1) \cdot \text{Enc}(\text{pk}, m_2; r_2) = \text{Enc}(\text{pk}, m_1 + m_2; r_1 + r_2)$ for $m_1, m_2 \in \mathcal{M}_{\text{Enc}}$ and $\mathcal{R}_{\text{Com}} \equiv \mathcal{M}_{\text{Enc}}$.
- **Public Inputs:** $\text{urs}_{zk} = (\text{urs}_{\text{com}}, \text{pk})$ where $(\text{urs}_{\text{com}}, \text{td}) \leftarrow \text{Com.Gen}(1^\lambda)$ and $(\text{pk}, \text{sk}) \leftarrow \text{PKE.Gen}(1^\lambda)$. Let $\ell = 8\lambda$.
- **Private Inputs:** V has input statement x for language \mathcal{L} . He runs an NP reduction to reduce it to an n -node input graph G for language \mathcal{L}_{Ham} . P has x and witness w for language \mathcal{L} . He runs the same NP reduction to obtain the n -node input graph G and a valid Hamiltonian Cycle ω .
- **Notations:** We denote prover algorithm as $P = (P_1, P_2)$ and verifier algorithm as $V = (V_1, V_2)$. For a graph G we denote the complement graph as \overline{G} .

Commit : $P_1(1^\lambda, 1^n)$
Repeat the following protocol for $i \in [\ell]$:

- Sample a random n -node cycle graph H^i .
- Commit to the adjacency matrix of H^i as follows:
 - For each edge j commit to 0, as $c_j^i = \text{Com}(0; r_j^i)$ with randomness r_j^i .
 - For each non-edge j commit to 1, as $c_j^i = \text{Com}(1; r_j^i)$ with randomness r_j^i .
 - For $j \in [n^2]$, the prover also encrypts the commitment randomness based on the bit committed as follows.
 - If $c_j^i = \text{Com}(0; r_j^i)$, then set $E_{j,0}^i = \text{Enc}(\text{pk}, r_j^i; s_j^i)$ with randomness s_j^i and sample $E_{j,1}^i \leftarrow \text{oEnc}(1^\lambda)$.
 - If $c_j^i = \text{Com}(1; r_j^i)$, then set $E_{j,1}^i = \text{Enc}(\text{pk}, r_j^i; s_j^i)$ with randomness s_j^i and sample $E_{j,0}^i \leftarrow \text{oEnc}(1^\lambda)$.

Samples an ℓ -bit random string $\mathbf{e}_P \leftarrow \{0, 1\}^\lambda$. Computes $T_e = \text{Enc}(\text{pk}, \mathbf{e}_P; d_e)$.

Construct the first message $\mathbf{a} = (T_e, \{c_j^i, E_{j,0}^i, E_{j,1}^i\}_{j \in [n^2], i \in [\ell]})$. Send \mathbf{a} to V. Set internal state $\text{st}_P = (\mathbf{e}_P, d_e, \{b_j^i, r_j^i, s_j^i\}_{j \in [n^2], i \in [\ell]})$.

Challenge : $V_1(1^\lambda)$
V samples $\mathbf{e}_V \leftarrow \{0, 1\}^\lambda$ and sends \mathbf{e}_V to P. Sets $\text{st}_V = \mathbf{e}_V$.

Response : $P_2((G, \omega), \mathbf{e}_V, \text{st}_P)$
Computes $\mathbf{e} = \mathbf{e}_P \oplus \mathbf{e}_V$ and repeats the following protocol for $i \in [\ell]$:

- Upon obtaining the statement graph G and Hamiltonian cycle ω as witness compute a permutation γ^i s.t. $H^i = \gamma^i(\omega)$.
- If $e^i = 0$, decommit to edges in H^i . P also opens to the commitment and encryption randomness, i.e. $z^i = \{j, b_j^i, r_j^i, s_j^i\}_{j \in H}$.
- If $e^i = 1$, reveal γ^i and decommit to non-edges in $\gamma^i(G)$. P also opens to the commitment and encryption randomness, i.e. $z^i = (\gamma^i, \{j, b_j^i, r_j^i, s_j^i\}_{j \in \overline{\gamma^i(G)}})$.

P sends the proof $\Gamma = (\mathbf{a}, \mathbf{e}, \mathbf{z}) = (\{a^1, \dots, a^\ell\}, \mathbf{e}, \{z^1, \dots, z^\ell\})$ and opening of T_e as (\mathbf{e}_P, d_e) .

Fig. 14. Adaptively Secure Rerandomizable ZK Protocol (cont.)

Verify : $V_2(G, \Gamma, \text{st}_V)$
V aborts if $T_e \neq \text{Enc}(\text{pk}, \mathbf{e}_P; d_e)$. Else, he sets $\mathbf{e} = \mathbf{e}_P \oplus \mathbf{e}_V$ and repeats the following protocol for $i \in [\ell]$:

- If $e^i = 0$, V outputs accept if for all $j \in H^i$, the following holds - $b_j^i = 1$, H^i is a cycle and $c_j^i = \text{Com}(1; r_j^i)$ where $E_{j,1}^i = \text{Enc}(\text{pk}, r_j^i; s_j^i)$.
- If $e^i = 1$, V outputs accept if for all $j \in \overline{\gamma^i(G)}$ the following holds - $b_j^i = 0$, $c_j^i = \text{Com}(0; r_j^i)$ and $E_{j,0}^i = \text{Enc}(\text{pk}, r_j^i; s_j^i)$.
- Else, V outputs reject.

Enc to simulate a proof. Our adaptive security proof also follows from the adaptive security of [CSW20b] by relying on the equivocal property of Com and oblivious ciphertext sampleability of Enc. \square

Next, we design a firewall for a tampered prover against a corrupt verifier. The code of the firewall is same for a tampered verifier against a corrupt prover. We denote the firewall as RF_{zk} . RF_{zk} rerandomizes the commitments and encryptions sent by the prover by relying on the additive homomorphic property of Com and Enc. The firewall also samples a random permutation γ' over the committed adjacency matrix and permutes the adjacency matrix according to it. He outputs the rerandomized commitments and encryptions in a permuted way such that it corresponds to $\gamma'(H)$. When P opens to H (for $e = 0$), the RF permutes the order of the openings according to γ' s.t. that they are consistent openings of $\gamma'(H)$. When P opens to the non-edges (for $e = 1$) corresponding to $\gamma(G)$, the RF permutes the order of the openings according to γ' s.t. that they correspond to non-edges in $\gamma'(\gamma(G))$. P sends the permutation γ and RF_{zk} rerandomizes it as $\gamma'(\gamma)$. The firewall rerandomizes the encryption T_e of prover's coin e_P and modifies the coin by an additive value e' . The sanitized encryption sent by the firewall (on behalf of prover) to the verifier is $\widehat{T}_e = T_e \cdot \text{Enc}(\text{pk}, e'; d'_e)$. When verifier sends his share e_V of the random coin, the firewall modifies the coin by an additive factor of e' . Thus, the final coin is $\widehat{e} = e_P + e_V + e'$ and it is ensured to be random since e' is random. We provide our firewall RF_{zk} for a tampered prover in Fig. 15. The same firewall works for a tampered verifier.

Theorem 10. *Let RF_{zk} (Fig. 15) be a reverse firewall for a tampered prover (resp. verifier) against a corrupt verifier (resp. prover) in Π_{zk} (Fig. 13, 14) and Π_{zk} implements \mathcal{F}_{ZK} functionality against adaptive corruption of parties. Let Com be a non-interactive equivocal commitment scheme in the urs model and PKE be an IND-CPA public key encryption scheme (where the public key is statistically close to a random string) with oblivious ciphertext sampleability. If Com and PKE are homomorphic with respect to the (addition) operation defined over the underlying spaces (i.e, the message space, randomness space) and the message space of PKE is same as randomness space of Com then the firewall RF_{zk} is transparent, functionality-maintaining and provides weak exfiltration resistance for a tampered prover (resp. verifier) against a corrupt verifier (resp. prover). The firewall also detects failures for all the parties.*

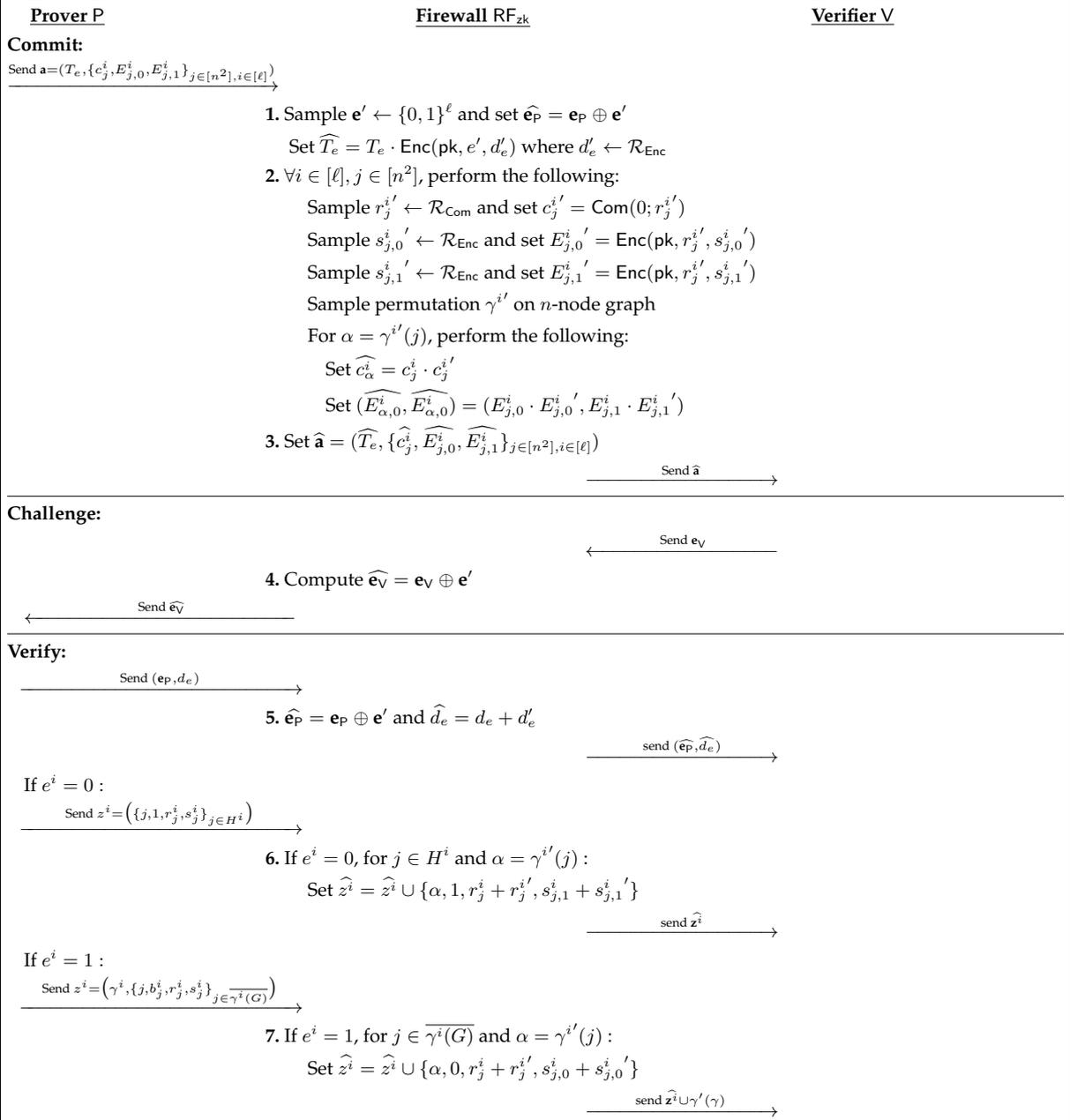
Proof Sketch. It can be observed that the firewall maintains functionality and it is transparent. When the verifier get adaptively corrupted, the Transform algorithm takes e_V as the state of V and e'_V as the state of RF_{zk} corresponding to verifier and returns $e_V \oplus e'_V$ as the state of the verifier composed with the firewall. When the prover get adaptively corrupted the Transform algorithm takes the openings of the committed adjacency matrix and the openings of the encryptions as the prover's state. It receives the randomness for the commitments and the encryptions, and the permutation from the firewall. It adds the state of the firewall to the state of the prover and permutes the adjacency matrix by the permutation provided by the firewall. This new state is returned as the state of the prover composed with the firewall.

If a corrupt prover provides an invalid response in the third round or opens its share of coin incorrectly then the firewall detects failure and sends a \perp to the verifier. Next, we briefly discuss that RF_{zk} provides weak exfiltration resistance for the weakly tampered parties.

- *Verifier is tampered* : The firewall rerandomizes a tampered verifier's challenge string by an additive offset e' . The rerandomized challenge is identically distributed to the rerandomization of an honestly generated challenge string. A weak ER adversary cannot distinguish between the two cases.
- *Prover is tampered* : A tampered prover can send a \mathbf{a} which is constructed using bad randomness. The firewall RF_{zk} rerandomizes it as $\widehat{\mathbf{a}}$. Due to the additive homomorphic property of

Fig. 15. Reverse firewall RF_{zk} for a tampered prover in Π_{zk} . The same firewall works for a tampered verifier

$\text{Com} = (\text{Gen}, \text{Com}, \text{Verify})$ is an additively homomorphic equivocal commitment scheme where $\text{Com}(\text{urs}, a_1; b_1) \cdot \text{Com}(\text{urs}, a_2; b_2) = \text{Com}(\text{urs}, a_1 + a_2; b_1 + b_2)$ for $a_1, a_2 \in \{0, 1\}^\lambda$ and $b_1, b_2 \in \mathcal{R}_{\text{Com}}$ respectively. $\text{PKE} = (\text{Gen}, \text{Enc}, \text{Dec})$ is an additively homomorphic IND-CPA encryption with oblivious ciphertext sampleability where $\text{Enc}(\text{pk}, m_1; r_1) \cdot \text{Enc}(\text{pk}, m_2; r_2) = \text{Enc}(\text{pk}, m_1 + m_2; r_1 + r_2)$ for $m_1, m_2 \in \mathcal{M}_{\text{Enc}}$ and $\mathcal{R}_{\text{Com}} \equiv \mathcal{M}_{\text{Enc}}, \mathcal{R}_{\text{Enc}}$ is the randomness space of Enc algorithm.



Com and PKE the distribution of \hat{a} is indistinguishable from an a which is generated using an honest implementation. If an ER adversary can distinguish the rerandomized messages of a tampered prover from the rerandomized messages of an honestly implemented prover then he can be used to break the hiding property of Com or the IND-CPA property of Enc.

We have proven that RF_{zk} provides weak exfiltration resistance and Thm. 9 proves that Π_{zk} implements \mathcal{F}_{ZK} functionality against adaptive corruption of parties. Combining the two results with Thm. 5 we prove that the firewall weakly preserves security against adaptive corruption of parties. The same argument holds true for multiple proofs where each proof is adaptively secure and possesses its own firewall providing weak exfiltration resistance for the parties. \square

From Thm. 10, and our implication from Thm. 5, we conclude weak security preservation; we have that $\Pi_{\text{zk}}^{\text{RF}_{\text{zk}}}$ securely implements \mathcal{F}_{ZK} in the presence of adaptive corruptions as summarized in Thm. 11 below.

Theorem 11. $\Pi_{\text{zk}}^{\text{RF}_{\text{zk}}}$ securely implements the \mathcal{F}_{ZK} functionality (Def. 12) in the urs model against adaptive corruption of parties, and in the presence of functionality-maintaining tampering of honest parties.

Instantiation from LWE. We define lattice parameters (n, m, q, N, χ) according to [GSW13] s.t LWE assumption holds. We consider the homomorphic trapdoor function of [GVW15] from LWE as our equivocal commitment scheme. The commitment scheme is fully homomorphic over the message space $\mathcal{M}_{\text{Com}} \in \{0, 1\}$ and it is additively homomorphic over the randomness space $\mathcal{R}_{\text{Com}} \in \mathbb{Z}_q^{m \times m}$. We consider the fully homomorphic encryption of GSW ([GSW13]), based on LWE assumption, as our PKE scheme. The encryption scheme is fully homomorphic over the message space $\mathcal{M}_{\text{Enc}} \in \mathbb{Z}_q$ and it is additively homomorphic over the randomness space $\mathcal{R}_{\text{Enc}} \in \{0, 1\}^{N \times m}$. Recall that, in Π_{zk} we need to encrypt the randomness of the commitment using our PKE. For each commitment $c = \text{Com}(m; r)$ to message m with randomness $r \leftarrow \mathbb{Z}_q^{m \times m}$ using the [GVW15] protocol, we encrypt r using m^2 GSW encryptions. The setup string urs_{com} consists of a random matrix $\mathbf{A}_{\text{Com}} \in \mathbb{Z}_q^{n \times m}$. The public key pk consists of a random matrix $\mathbf{A}_{\text{Enc}} \in \mathbb{Z}_q^{m \times (n+1)}$ during the protocol execution. This yields Π_{zk} in the uniform random string model based on LWE assumption.

7.3 Adaptively-Secure MPC in the urs_{mpc} model

We present our actively-secure protocol Π_{mpc} which withstands adaptive corruption of parties in Fig. 16, 17. Adaptive security is achieved by the adaptive security of the underlying primitives and subprotocols – ZK, augmented coin-tossing, semi-honest MPC protocol and commitments.

Theorem 12. Assuming Com is an adaptively secure commitment in the urs_{com} model, $\Pi_{\text{a-coin}}$ securely implements the augmented coin-tossing functionality against adaptive corruption of parties in the $\text{urs}_{\text{a-coin}}$ model, Π_{zk} securely implements the \mathcal{F}_{ZK} functionality against adaptive corruption of parties in the urs_{zk} model and $\Pi_{\text{sh-mpc}}$ is an adaptively-secure semi-honest MPC protocol in the $\text{urs}_{\text{sh-mpc}}$ model, Π_{mpc} is an actively secure MPC protocol that withstands adaptive corruption of parties in the $\text{urs}_{\text{mpc}} = (\text{urs}_{\text{com}}, \text{urs}_{\text{zk}}, \text{urs}_{\text{a-coin}}, \text{urs}_{\text{sh-mpc}})$ model.

Proof. We prove our security by providing an adaptive simulator Sim_{mpc} in Fig. 18, 19, 20 for Π_{mpc} . The semi-honest mpc protocol has a simulator $\text{Sim}_{\text{sh-mpc}}$ which simulates the protocol

Fig. 16. Adaptively secure Multi-party Protocol Π_{mpc} in the urs model

- **Primitives:** (Gen, Com, Verify) is an adaptively secure non-interactive commitment scheme and $\Pi_{\text{zk}} = (\text{Gen}, \text{P}_1, \text{V}_1, \text{P}_2, \text{V}_2)$ is an adaptively secure rerandomizable input-delayed protocol implementing \mathcal{F}_{ZK} in the urs_{zk} model.
- **Subprotocols:** Adaptively secure augmented coin-tossing protocol $\Pi_{\text{a-coin}}$ in the $\text{urs}_{\text{a-coin}}$ model and adaptively secure semi-honest k -round MPC protocol $\Pi_{\text{sh-mpc}}$ in the $\text{urs}_{\text{sh-mpc}}$ model. Each invocation of $\Pi_{\text{a-coin}}$ generates λ for initiating party.
- **Inputs:** Each party gets $\text{urs}_{\text{mpc}} = (\text{urs}_{\text{com}}, \text{urs}_{\text{zk}}, \text{urs}_{\text{a-coin}}, \text{urs}_{\text{sh-mpc}})$ and security parameter 1^λ . Party P_i has private input x_i for $i \in [n]$.
- **Output:** Parties compute the ideal functionality \mathcal{F}_{mpc} and output $y = \mathcal{F}_{\text{mpc}}(x_1, x_2, \dots, x_n) = f(x_1, x_2, \dots, x_n)$.
- **Notations:** Let $\mathcal{T}^{<T} = \bigcup_{t \in [T-1]} \{\mathcal{T}_i^t\}_{i \in [n]}$ denote the entire transcript of $\Pi_{\text{sh-mpc}}$ until the end of round $T - 1$ for $0 < T \leq k$. We assume $\mathcal{T}^0 = \perp$. NMF is the next message function of $\Pi_{\text{sh-mpc}}$ for P_i computing $\text{NMF}(\mathcal{T}^{<T}, x_i, r_i) = \mathcal{T}_i^T$. Let $N\lambda = \text{no. of random bits required by } P_i \text{ to commit } |x_i| \text{ bits} + \text{no. of random bits required by } P_i \text{ to compute } \Pi_{\text{sh-mpc}}$. $\gamma_{\text{inp}, i, j}^\ell$ denotes the ℓ -th ($\ell \in [3]$) round message in the ZK proof for \mathcal{R}_{inp} where P_i is the prover and P_j is the verifier. Similarly, $\gamma_{t, i, j}^\ell$ (where $t \in [k]$) denotes the ℓ -th ($\ell \in [3]$) round message in the ZK proof for \mathcal{R}_{mpc} where P_i is the prover and P_j is the verifier.
- **Relations:** Let $\mathcal{R}_{\text{inp}}((c, c^{\text{inp}}), (x, r, s)) = 1$ iff $(c = \text{Com}(x; r) \wedge c^{\text{inp}} = \text{Com}(r; s))$. Let $\mathcal{R}_{\text{mpc}}((\mathcal{T}, \mathcal{T}', c, c^{\text{inp}}, c'), (x, r, s, r', s')) = 1$ iff $(c = \text{Com}(x; r) \wedge c^{\text{inp}} = \text{Com}(r; s) \wedge c = \text{Com}(r'; s') \wedge \text{NMF}(\mathcal{T}', x, r') = \mathcal{T})$.

Offline Phase:

The parties run the following protocols in parallel:

- For $i \in [n]$, Party P_i invokes $\Pi_{\text{a-coin}}$ N times in parallel as the initiating party to obtain randomness for input commitment - $(r_i^{\text{inp}}, s_i^{\text{inp}})$ s.t. $c_i^{\text{inp}} = \text{Com}(r_i^{\text{inp}}; s_i^{\text{inp}})$, and randomness for MPC protocol $(r_i^{\text{mpc}}, s_i^{\text{mpc}})$ s.t. $c_i^{\text{mpc}} = \text{Com}(r_i^{\text{mpc}}; s_i^{\text{mpc}}) = \{r_i^{\text{mpc}}[t], c_i^{\text{mpc}}[t]\}_{t \in [k]}$. Every party obtains c_i^{inp} and c_i^{mpc} .
- For $i \in [n], j \in [n] \setminus i$, Party P_i runs $\Pi_{\text{zk}}.\text{P}_1(1^\lambda, 1^{|\mathcal{R}_{\text{inp}}|})$ as prover with P_j as verifier to obtain $\gamma_{\text{inp}, i, j}^1$. Upon obtaining $\gamma_{\text{inp}, i, j}^1$, P_j runs $\text{V}_2 = \Pi_{\text{zk}}(1^\lambda)$ on $\gamma_{\text{inp}, i, j}^1$ to obtain $\gamma_{\text{inp}, i, j}^2$. P_i and P_j obtain $(\gamma_{\text{inp}, i, j}^1, \gamma_{\text{inp}, i, j}^2)$. P_i aborts if $\gamma_{\text{inp}, i, j}^2$ is invalid.
- For $i \in [n], j \in [n] \setminus i, t \in [k]$, Party P_i runs $\Pi_{\text{zk}}.\text{P}_1(1^\lambda, 1^{|\mathcal{R}_{\text{mpc}}|})$ as prover with P_j as verifier to obtain $\gamma_{t, i, j}^1$. Upon obtaining $\gamma_{t, i, j}^1$, P_j runs $\text{V}_2 = \Pi_{\text{zk}}(1^\lambda)$ on $\gamma_{t, i, j}^1$ to obtain $\gamma_{t, i, j}^2$. P_i and P_j obtain $(\gamma_{t, i, j}^1, \gamma_{t, i, j}^2)$. P_i aborts if $\gamma_{t, i, j}^2$ is invalid.

Online Phase:

Each party P_i (for $i \in [n]$) performs the following :

Input Commitment Phase:

- Each party P_i commits to his input x_i as $c_i = \text{Com}(x_i; r_i^{\text{inp}})$ and broadcasts c_i .
- For each $j \in [n] \setminus i$, party P_i proves honest computation of c_i using the committed randomness. P_i computes proof $\gamma_{\text{inp}, i, j}^3 = \Pi_{\text{zk}}.\text{P}_2$ for $\mathcal{R}_{\text{inp}}((c_i, c_i^{\text{inp}}), (x_i, r_i^{\text{inp}}, s_i^{\text{inp}}))$ on $(\gamma_{\text{inp}, i, j}^1, \gamma_{\text{inp}, i, j}^2)$. P_i sends $\Gamma_{\text{inp}} = (\gamma_{\text{inp}, i, j}^1, \gamma_{\text{inp}, i, j}^2, \gamma_{\text{inp}, i, j}^3)$ to P_j .

Fig. 17. Adaptively-Secure Multi-party Protocol in the urs model(cont.)

<p><i>Local Computation at the end of Input Commitment Phase:</i> After receiving the n commitments party P_i aborts if $\exists j \in [n]$ s.t. $\Pi_{\text{zk}}.\mathcal{V}_2(\Gamma_{\text{inp},j,i}) = 0$.</p> <p>.....</p> <p>Round $1 \leq t \leq k$:</p> <ul style="list-style-type: none"> - If any party aborts at the end of round $t - 1$ then abort. - P_i computes the t-th round message of the MPC protocol as $\mathcal{T}^t = \text{NMF}(\mathcal{T}^{<t}, x_i, r_i^{\text{mpc}}[t])$. P_i broadcasts \mathcal{T}_i^t. - For each $j \in [n] \setminus i$, party P_i proves honest computation of \mathcal{T}_i^t to P_j using the committed input x_i and committed randomness $r_i^{\text{mpc}}[t]$. P_i computes proof $\gamma_{t,i,j}^3 = \Pi_{\text{zk}}.\text{P}_2$ for $\mathcal{R}_{\text{mpc}}((\mathcal{T}_i^t, \mathcal{T}^{<t}, c_i, c_i^{\text{inp}}, c_i^{\text{mpc}}[t]), (x_i, r_i^{\text{inp}}, s_i^{\text{inp}}, r_i^{\text{mpc}}[t], s_i^{\text{mpc}}[t]))$ on $(\gamma_{t,i,j}^1, \gamma_{t,i,j}^2)$. P_i sends $\Gamma_{t,i,j} = (\gamma_{t,i,j}^1, \gamma_{t,i,j}^2, \gamma_{t,i,j}^3)$ to P_j. <p><i>Local Computation at the end of Round t:</i></p> <ul style="list-style-type: none"> - Party P_i aborts if $\exists j \in [n] \setminus i$ s.t. $\Pi_{\text{zk}}.\mathcal{V}_2(\Gamma_{t,j,i}) = \text{reject}$. - For $i \in [n]$, P_i sets $\mathcal{T}^{<t+1} = \mathcal{T}^{<t} \cup \{\mathcal{T}_j^t\}_{j \in [n]}$ and continues to next round. <p><i>Output Computation for Party $\{P_i\}_{i \in [n]}$:</i></p> <ul style="list-style-type: none"> - If any party aborts at the end of round k then abort. - P_i sets $\mathcal{T}^{\leq k} = \mathcal{T}^{<k} \cup \{\mathcal{T}_j^k\}_{j \in [n]}$ and outputs $y = \text{NMF}(\mathcal{T}^{\leq k}, x_i, r_i^{\text{mpc}}[k])$.
--

steps of $\Pi_{\text{sh-mpc}}$ given the output and the input of the corrupt parties. The augmented coin-tossing protocol $\Pi_{\text{a-coin}}$ has an adaptive simulator $\text{Sim}_{\text{a-coin}}$ which extracts the secret random coins from a corrupt initiating party. $\text{Sim}_{\text{a-coin}}$ also allows a simulated initiating party to equivocate his committed coins. Sim_{zk} computes simulated proofs on behalf of an honest prover and provides consistent randomness corresponding to a witness when the prover gets corrupted. Sim_{zk} also extracts a valid witness from an accepting proof which is constructed by a malicious prover.

On a high level, Sim_{mpc} invokes $\text{Sim}_{\text{a-coin}}$ to obtain simulated randomness and simulated commitments of randomness on behalf of initiating parties who are honest. These randomness are used for the input commitments and the MPC protocol. Sim_{mpc} invokes Sim_{zk} to obtain simulated messages for the first two rounds of each ZK protocol where the prover is an honest party participating in the MPC protocol. These are performed in the offline phase. In the online phase, Sim_{mpc} invokes Sim_{Com} to obtain simulated commitments on behalf of the honest parties and he invokes Sim_{zk} on the simulated commitments for relation \mathcal{R}_{inp} to obtain simulated proof for input commitments. These commitments and proofs are broadcasted as part of the input commitment phase. Sim_{mpc} obtains the input commitments and proofs constructed by malicious parties. Sim_{mpc} invokes Sim_{Com} to extract the corresponding inputs. It also invokes $\text{Sim}_{\text{a-coin}}$ to extract the committed randomness of the dishonest parties. The same inputs and randomness should have been used by the corrupt parties to construct ZK proofs. So, Sim_{mpc} also extracts the witness from the ZK proofs and verifies their consistency with the previously extracted values. If all the checks pass, then the extracted inputs are correct and Sim_{mpc} invokes the ideal functionality with the extracted inputs of the corrupt parties to obtain output y . Now, Sim_{mpc} invokes $\text{Sim}_{\text{sh-mpc}}$ with the extracted inputs and output to obtain simulated MPC messages. If a party P_i gets adaptively corrupted then Sim_{mpc} obtains the party's input x_i . It invokes $\text{Sim}_{\text{sh-mpc}}$ with x_i to obtain randomness r_i^{mpc} . Sim_{mpc} invokes Sim_{Com} with x_i to obtain the randomness r_i^{inp} . Sim_{mpc} invokes $\text{Sim}_{\text{a-coin}}$ on r_i^{inp} and r_i^{mpc} to obtain the randomness values s_i^{inp} and s_i^{mpc} . Sim_{mpc} invokes Sim_{zk} with input witnesses $(x_i, r_i^{\text{inp}}, s_i^{\text{inp}}, r_i^{\text{mpc}}, s_i^{\text{mpc}})$ and $(x_i, r_i^{\text{inp}}, s_i^{\text{inp}})$ for proofs constructed by P_i (as prover) to obtain simulated prover states which are consistent with the witness. Sim_{mpc} returns $(x_i, r_i^{\text{inp}}, s_i^{\text{inp}}, r_i^{\text{mpc}}, s_i^{\text{mpc}})$ and the simulated prover states as the view of P_i . This view consists of the adversary's view corresponding to corrupt party P_i . The final view of the adversary consists of all the corrupt parties' views.

Our formal simulator is provided in Fig. 18, 19, 20 for completeness and we prove that the real world adversary view is indistinguishable from the ideal world adversary view as follows:

- Hyb_0 : This is the real world execution of the protocol.
- Hyb_1 : Same as Hyb_0 except the ZK proofs are simulated by invoking Sim_{zk} . Sim_{zk} is also used to extract $(x_j, r_j^{\text{inp}}, s_j^{\text{inp}}, r_j^{\text{mpc}}[t], s_j^{\text{mpc}}[t])$ and $(x_j, r_j^{\text{inp}}, s_j^{\text{inp}})$ from maliciously constructed proofs $\Gamma_{t,j,i}$ and $\Gamma_{\text{inp},j,i}$ for $j \in C$ and $i \in H$. If an honest party P_i gets adaptively corrupted then his simulated proof randomness is obtained by invoking Sim_{zk} on the simulated proof. Sim_{mpc} aborts if Sim_{zk} fails to produce simulated proofs for honest parties or open those simulated proofs to witnesses when the prover gets adaptively corrupted. Sim_{mpc} also aborts if Sim_{zk} fails to extract from maliciously constructed proof.

Indistinguishability follows due to adaptive security of Π_{zk} .

- Hyb_2 : Same as Hyb_1 except the augmented coin-tossing protocol for an honest initiating party is simulated by invoking $\text{Sim}_{\text{a-coin}}$. For a corrupt initiating party the committed coins are extracted by invoking $\text{Sim}_{\text{a-coin}}$. If the honest party gets adaptively corrupted then his simulated coins are opened consistent to the required coins by invoking $\text{Sim}_{\text{a-coin}}$. Sim_{mpc} aborts if $\text{Sim}_{\text{a-coin}}$ fails to produce simulated committed coins for honest initiating parties or open those committed coins to required coins when the initiating party gets adaptively corrupted. Sim_{mpc} also aborts if $\text{Sim}_{\text{a-coin}}$ fails to extract from committed coins where the initiating party is corrupt.

Indistinguishability follows due to adaptive security of $\Pi_{\text{a-coin}}$.

- Hyb_3 : Same as Hyb_2 except the input commitment for an honest party is simulated by invoking Sim_{Com} . For a corrupt party the committed inputs are extracted by invoking Sim_{Com} . If the honest party gets adaptively corrupted then his simulated commitments consistently to his input by invoking Sim_{Com} . Sim_{mpc} aborts if Sim_{Com} fails to produce simulated commitments for honest parties or open those commitments to the party's input when the party gets adaptively corrupted. Sim_{mpc} also aborts if Sim_{Com} fails to extract from input commitments constructed by malicious parties.

Indistinguishability follows due to adaptive security of Com .

- Hyb_4 : Same as Hyb_3 except Sim_{mpc} invokes the ideal functionality with the extracted inputs of corrupt parties to obtain the output y . Sim_{mpc} invokes $\text{Sim}_{\text{sh-mpc}}$ with the input and output of the corrupt parties to obtain simulated protocol messages on behalf of the honest parties. If an honest party gets adaptively corrupted and Sim_{mpc} obtains the party's input then Sim_{mpc} invokes $\text{Sim}_{\text{sh-mpc}}$ with the party's input to obtain the party's view. Sim_{mpc} aborts if $\text{Sim}_{\text{sh-mpc}}$ fails to produce a consistent view.

Indistinguishability follows due to adaptive security of $\text{Sim}_{\text{sh-mpc}}$.

□

Next, we consider a reverse firewall $\text{RF}_{\text{mpc}}^i = (\text{RF}_{\text{zk}}^i, \text{RF}_{\text{a-coin}}^i)$ to be the firewall for P_i in Π_{mpc} . RF_{mpc}^i is obtained by first applying RF_{zk}^i to the messages of Π_{zk} phase of Π_{mpc} , followed by application of $\text{RF}_{\text{a-coin}}^i$ to the messages in the $\Pi_{\text{a-coin}}$ phase, if RF_{zk}^i did not output \perp . We show that RF_{mpc}^i provides weak exfiltration resistance for party P_i in Π_{mpc} .

Theorem 13. *Let Com be an adaptively secure commitment in the urs_{com} model, $\Pi_{\text{a-coin}}$ securely implement the augmented coin tossing functionality against adaptive corruption of parties in the $\text{urs}_{\text{a-coin}}$ model, Π_{zk} securely implement the \mathcal{F}_{zk} functionality against adaptive corruption of parties in the urs_{zk} model, and $\Pi_{\text{sh-mpc}}$ be an adaptively-secure semi-honest MPC protocol in the $\text{urs}_{\text{sh-mpc}}$ model. Let RF_{zk}^i*

Fig. 18. Adaptive Simulator Sim_{mpc} for Π_{mpc}

Sim_{mpc}

- **Public Inputs:** Each party receives $\text{urs}_{\text{mpc}} = (\text{urs}_{\text{com}}, \text{urs}_{\text{zk}}, \text{urs}_{\text{a-coin}}, \text{urs}_{\text{sh-mpc}})$ as setup string and security parameter 1^λ .
- **Trapdoors:** Trapdoor for $\text{urs}_{\text{mpc}} = (\text{urs}_{\text{com}}, \text{urs}_{\text{zk}}, \text{urs}_{\text{a-coin}}, \text{urs}_{\text{sh-mpc}})$ is denoted as $\text{td}_{\text{mpc}} = (\text{td}_{\text{com}}, \text{td}_{\text{zk}}, \text{td}_{\text{a-coin}}, \text{td}_{\text{sh-mpc}})$.
- **Simulators:** Adaptive simulator Sim_{Com} for Com , adaptive simulator Sim_{zk} for Π_{zk} and adaptive simulator $\text{Sim}_{\text{a-coin}}$ for $\Pi_{\text{a-coin}}$ and adaptive simulator $\text{Sim}_{\text{sh-mpc}}$ for $\Pi_{\text{sh-mpc}}$ respectively. $\text{Sim}_{\text{Com}}, \text{Sim}_{\text{zk}}, \text{Sim}_{\text{a-coin}}$ and $\text{Sim}_{\text{sh-mpc}}$ knows the trapdoors $\text{td}_{\text{com}}, \text{td}_{\text{zk}}, \text{td}_{\text{a-coin}}$ and $\text{td}_{\text{sh-mpc}}$ respectively.
- **Ideal Functionality:** Ideal functionality \mathcal{F}_{mpc} computes the function $f(x_1, x_2, \dots, x_n) = y$ over the inputs of the parties.
- **Notations:** C denotes the current set of corrupt parties and H denotes the current set of honest parties. $[a, b]$ denotes the set $(a, a + 1, \dots, b)$ where $a < b$. $\gamma_{\text{inp}, i, j}^\ell$ denotes the ℓ -th ($\ell \in [3]$) round message in the ZK proof for \mathcal{R}_{inp} where P_i is the prover and P_j is the verifier. Similarly, $\gamma_{t, i, j}^\ell$ (where $t \in [k]$) denotes the ℓ -th ($\ell \in [3]$) round message in the ZK proof for \mathcal{R}_{mpc} where P_i is the prover and P_j is the verifier.
- **Relations:** Let $\mathcal{R}_{\text{inp}}((c, c^{\text{inp}}), (x, r, s)) = 1$ iff $(c = \text{Com}(x; r) \wedge c^{\text{inp}} = \text{Com}(r; s))$. Let $\mathcal{R}_{\text{mpc}}((\mathcal{T}, \mathcal{T}', c, c^{\text{inp}}, c'), (x, r, s, r', s')) = 1$ iff $(c = \text{Com}(x; r) \wedge c^{\text{inp}} = \text{Com}(r; s) \wedge c = \text{Com}(r'; s') \wedge \text{NMF}(\mathcal{T}', x, r') = \mathcal{T})$.

Offline Phase:
The simulator Sim_{mpc} simulates on behalf of the honest parties as follows:

- The simulator algorithm runs $\text{Sim}_{\text{a-coin}}$ on behalf of honest parties P_i for $i \in H$ for N times to obtain simulated commitment and randomness pairs - $(c_i^{\text{inp}}, r_i^{\text{inp}'})$ and $(c_i^{\text{mpc}}, r_i^{\text{mpc}'})$, and simulated states $s_i^{\text{inp}'}$ and $s_i^{\text{mpc}'}$ respectively. Every party obtains c_j^{inp} and c_j^{mpc} for $j \in [n]$.
- The simulator algorithm runs Sim_{zk} on $(1^\lambda, 1^{|\mathcal{R}_{\text{inp}}|})$ as honest prover on behalf of honest parties P_i for $i \in H, j \in [n] \setminus$ for k times to obtain simulated proofs $(\gamma_{\text{inp}, i, j}^1, \gamma_{\text{inp}, i, j}^2)$ and simulated states $\text{st}'_{\text{inp}, i, j}$ where $t \in [k]$. Every pair of P_i and P_j obtain $(\gamma_{\text{inp}, i, j}^1, \gamma_{\text{inp}, i, j}^2)$. Sim_{mpc} aborts if Sim_{zk} aborts.
- The simulator algorithm runs Sim_{zk} on $(1^\lambda, 1^{|\mathcal{R}_{\text{mpc}}|})$ as honest prover on behalf of honest parties P_i for $i \in H, j \in [n] \setminus$ for k times to obtain simulated proofs $(\gamma_{t, i, j}^1, \gamma_{t, i, j}^2)$ and simulated states $\text{st}'_{t, i, j}$ where $t \in [k]$. Every pair of P_i and P_j obtain $(\gamma_{t, i, j}^1, \gamma_{t, i, j}^2)$ for $t \in [k]$. Sim_{mpc} aborts if Sim_{zk} aborts.

Adaptive Corruption of Party P_i :
 Sim_{mpc} obtains x_i as input of P_i . Sim_{mpc} samples random strings for r_i^{inp} and r_i^{mpc} . It obtains s_i^{inp} and s_i^{mpc} as the commitment randomness corresponding to r_i^{inp} and r_i^{mpc} by invoking $\text{Sim}_{\text{a-coin}}$ on $(c_i^{\text{inp}}, r_i^{\text{inp}}, r_i^{\text{inp}'}, s_i^{\text{inp}'})$ and $(c_i^{\text{mpc}}, r_i^{\text{mpc}}, r_i^{\text{mpc}'}, s_i^{\text{mpc}'})$ respectively. Sim_{mpc} invokes Sim_{zk} on $((\gamma_{t, i, j}^1, \gamma_{t, i, j}^2), \perp, \text{st}'_{t, i, j})$ to obtain $\text{st}_{t, i, j}$ for $j \in [n] \setminus i, t \in [k]$. Sim_{mpc} invokes Sim_{zk} on $((\gamma_{\text{inp}, i, j}^1, \gamma_{\text{inp}, i, j}^2), \perp, \text{st}'_{\text{inp}, i, j})$ to obtain $\text{st}_{\text{inp}, i, j}$ for $j \in [n] \setminus i, t \in [k]$. Sim_{mpc} returns $(x_i, r_i^{\text{inp}}, s_i^{\text{inp}}, r_i^{\text{mpc}}, s_i^{\text{mpc}}, \{\text{st}_{t, i, j}\}_{j \in [n] \setminus i, t \in [k]}, \{\text{st}_{\text{inp}, i, j}\}_{j \in [n] \setminus i})$ as P_i 's view.

Fig. 19. Adaptive Simulator for Π_{mpc} (cont.)

Input Commitment Phase:

- Sim_{mpc} obtains simulated commitments and simulated openings as $(c_i, x'_i, r_i^{\text{inp}'})$ by invoking Sim_{Com} . Sim_{mpc} broadcasts c_i on behalf of P_i .
- For each $j \in [n] \setminus i$, Sim_{mpc} invokes Sim_{zk} on statement (c_i, c_i^{inp}) and $(\gamma_{\text{inp},i,j}^1, \gamma_{\text{inp},i,j}^2)$ for relation \mathcal{R}_{inp} to obtain simulated proofs $\Gamma_{\text{inp},i,j} = (\gamma_{\text{inp},i,j}^1, \gamma_{\text{inp},i,j}^2, \gamma_{\text{inp},i,j}^3)$ and simulated prover state $\text{st}'_{\text{inp},i,j}$. Sim_{mpc} sends $\Gamma_{\text{inp},i,j}$ to P_j on behalf of P_i .

Local Computation by Sim_{mpc} at the end of Input Commitment Phase:

- For $j \in C$, Sim_{mpc} extracts x_j by invoking Sim_{Com} on c_j and aborts if $x_j = \perp$.
- For $j \in C$, Sim_{mpc} extracts $(r_j^{\text{inp}}, s_j^{\text{inp}})$ by invoking $\text{Sim}_{\text{a-coin}}$ on c_j^{inp} and c_j^{mpc} and aborts if $r_j^{\text{inp}} = \perp$ or $s_j^{\text{inp}} = \perp$.
- For $j \in C$, Sim_{mpc} extracts P_j 's proof witness - $(x'_j, r_j^{\text{inp}'}, s_j^{\text{inp}'})$ by invoking Sim_{zk} on $\Gamma_{\text{inp},j,i}$ for every $i \in H$. Sim_{mpc} aborts if $x_j \neq x'_j, r_j^{\text{inp}} \neq r_j^{\text{inp}'}$ or $s_j^{\text{inp}} \neq s_j^{\text{inp}'}$.
- Sim_{mpc} sets $\mathcal{T}^{<2} = \mathcal{T}^1 = \{\mathcal{T}_j^1\}_{j \in [n]}$ for every $\{P_i\}_{i \in H}$.
- Sim_{mpc} invokes ideal functionality \mathcal{F}_{mpc} with inputs $\{x_j\}_{j \in C}$ to obtain output y .

Adaptive Corruption of Party P_i :

Sim_{mpc} obtains x_i as input of P_i . Sim_{mpc} invokes Sim_{Com} with $(c_i, x_i, x'_i, r_i^{\text{inp}'})$ to obtain randomness r_i^{inp} . Sim_{mpc} invokes $\text{Sim}_{\text{a-coin}}$ with inputs $(c_i^{\text{inp}}, r_i^{\text{inp}}, r_i^{\text{inp}'}, s_i^{\text{inp}'})$ to obtain s_i^{inp} . Sim_{mpc} invokes Sim_{zk} on $(\Gamma_{\text{inp},i,j})$ to obtain $\text{st}_{\text{inp},i,j}$. Sim_{mpc} invokes Sim_{zk} on $((\gamma_{t,i,j}^1, \gamma_{t,i,j}^2), \perp, \text{st}'_{t,i,j})$ to obtain $\text{st}_{t,i,j}$ for $j \in [n] \setminus i, t \in [k]$. Sim_{mpc} samples random string for r_i^{mpc} and obtains s_i^{mpc} as the commitment randomness corresponding to r_i^{mpc} by invoking $\text{Sim}_{\text{a-coin}}$ on $(c_i^{\text{mpc}}, r_i^{\text{mpc}}, r_i^{\text{mpc}'}, s_i^{\text{mpc}'})$. Sim_{mpc} returns $(x_i, r_i^{\text{inp}}, s_i^{\text{inp}}, r_i^{\text{mpc}}, s_i^{\text{mpc}}, \{\text{st}_{t,i,j}\}_{j \in [n] \setminus i, t \in [k]})$ as P_i 's view.

.....
Round $1 \leq t \leq k$:

- Sim_{mpc} invokes $\text{Sim}_{\text{sh-mpc}}$ on $(\{x_j\}_{j \in C}, y, \mathcal{T}^{<t}, \{r_i^{\text{mpc}''}[1, t-1]\}_{i \in H})$ to obtain simulated t -th round message for $\{P_i\}_{i \in H}$ as $\{\mathcal{T}_i^t\}_{i \in H}$ and simulated randomness $\{r_i^{\text{mpc}''}[t]\}_{i \in H}$. For $i \in H$, Sim_{mpc} broadcasts \mathcal{T}_i^t on behalf of P_i .
- Sim_{mpc} simulates proofs for every honest party P_i (where $i \in H$) as follows: For $j \in [n] \setminus i$ Sim_{mpc} invokes Sim_{zk} on statement $(\mathcal{T}_i^t, \mathcal{T}^{<t}, c_i, c_i^{\text{inp}}, c_i^{\text{mpc}}[t])$ for relation \mathcal{R}_{mpc} to obtain simulated proofs $\Gamma_{t,i,j} = (\gamma_{t,i,j}^1, \gamma_{t,i,j}^2, \gamma_{t,i,j}^3)$ and simulated prover state $\text{st}'_{t,i,j}$. Sim_{mpc} sends $\Gamma_{t,i,j}$ to P_j on behalf of P_i .

Local Computation by Sim_{mpc} at the end of Round t :

- For $j \in C$, Sim_{mpc} extracts $(r_j^{\text{mpc}}[t], s_j^{\text{mpc}}[t])$ by invoking $\text{Sim}_{\text{a-coin}}$ on $c_j^{\text{mpc}}[t]$ and aborts if $r_j^{\text{mpc}}[t] = \perp$ or $s_j^{\text{mpc}}[t] = \perp$.
- For $j \in C$, Sim_{mpc} extracts P_j 's proof witness - $(x'_j, r_j^{\text{inp}'}, s_j^{\text{inp}'}, r_j^{\text{mpc}}[t]', s_j^{\text{mpc}}[t]')$ by invoking Sim_{zk} on $\Gamma_{t,j,i}$ for every $i \in H$. Sim_{mpc} aborts if $x_j \neq x'_j, r_j^{\text{inp}} \neq r_j^{\text{inp}'}, s_j^{\text{inp}} \neq s_j^{\text{inp}'}, r_j^{\text{mpc}}[t] \neq r_j^{\text{mpc}}[t]'$ or $s_j^{\text{mpc}}[t] \neq s_j^{\text{mpc}}[t]'$.
- Else, Sim_{mpc} sets $\mathcal{T}^{<t+1} = \mathcal{T}^{<t} \cup \{\mathcal{T}_j^t\}_{j \in [n]}$ for every $\{P_i\}_{i \in H}$ and continues to next round.

Fig. 20. Adaptive Simulator for Π_{mpc} (cont.)

<p>Adaptive Corruption of Party P_i:</p> <p>Sim_{mpc} obtains x_i as input of P_i. Sim_{mpc} invokes $\text{Sim}_{\text{sh-mpc}}$ with input $(\{x_j\}_{j \in C}, y, \mathcal{T}^{\leq t}, x_i, r_i^{\text{mpc}''}[1, t])$ to obtain randomness $r_i^{\text{mpc}}[1, t]$ for the MPC protocol. Sim_{mpc} sets $r_i^{\text{mpc}}[t+1, k]$ randomly. Sim_{mpc} invokes $\text{Sim}_{\text{a-coin}}$ with input $(c_i^{\text{mpc}}, r_i^{\text{mpc}}, r_i^{\text{mpc}'}, s_i^{\text{mpc}'})$ to obtain randomness s_i^{mpc}. Sim_{mpc} invokes Sim_{zk} on $(\Gamma_{u,i,j}, (x_i, r_i^{\text{inp}}, s_i^{\text{inp}}, r_i^{\text{mpc}}[u], s_i^{\text{mpc}}[u]), \text{st}'_{u,i,j})$ to obtain $\text{st}_{u,i,j}$ for $j \in [n] \setminus i, u \in [1, t]$. Sim_{mpc} invokes Sim_{zk} on $(\gamma_{u,i,j}^1, \gamma_{u,i,j}^2, \perp, \text{st}'_{u,i,j})$ to obtain $\text{st}_{u,i,j}$ for $j \in [n] \setminus i, u \in [t+1, k]$. Sim_{mpc} returns $(x_i, r_i^{\text{inp}}, s_i^{\text{inp}}, r_i^{\text{mpc}}, s_i^{\text{mpc}}, \{\text{st}_{u,i,j}\}_{j \in [n] \setminus i, u \in [k]})$ as P_i's view.</p> <p>.....</p> <p>Output Computation for Party $\{P_i\}_{i \in H}$:</p> <ul style="list-style-type: none"> – If any party aborts at the end of round k then abort. – Else, Sim_{mpc} sets $\mathcal{T}^{\leq k} = \mathcal{T}^{< k} \cup \{\mathcal{T}_j^t\}_{j \in [n]}$ and outputs y. <p>Adaptive Corruption of Party P_i:</p> <p>Sim_{mpc} obtains x_i as input of P_i. Sim_{mpc} invokes $\text{Sim}_{\text{sh-mpc}}$ with input $(\{x_j\}_{j \in C}, y, \mathcal{T}^{\leq k}, x_i, r_i^{\text{mpc}''})$ to obtain randomness r_i^{mpc} for the MPC protocol. Sim_{mpc} invokes $\text{Sim}_{\text{a-coin}}$ with input $(c_i^{\text{mpc}}, r_i^{\text{mpc}}, r_i^{\text{mpc}'}, s_i^{\text{mpc}'})$ to obtain randomness s_i^{mpc}. Sim_{mpc} invokes Sim_{zk} on $(\Gamma_{u,i,j}, (x_i, r_i^{\text{inp}}, s_i^{\text{inp}}, r_i^{\text{mpc}}[u], s_i^{\text{mpc}}[u]), \text{st}'_{u,i,j})$ to obtain $\text{st}_{u,i,j}$ for $j \in [n] \setminus i, u \in [1, k]$. Sim_{mpc} returns $(x_i, r_i^{\text{inp}}, s_i^{\text{inp}}, r_i^{\text{mpc}}, s_i^{\text{mpc}}, \{\text{st}_{u,i,j}\}_{j \in [n] \setminus i, u \in [k]})$ as P_i's view.</p>

and $\text{RF}_{\text{a-coin}}^i$ be transparent, functionality-maintaining, and weakly exfiltration resistant for P_i in Π_{zk} and $\Pi_{\text{a-coin}}$ respectively. Then, RF_{mpc}^i is a transparent, functionality-maintaining, weakly exfiltration resistant firewall for P_i in Π_{mpc} .

Proof. The firewall RF_{mpc} is functionality maintaining since the individual firewalls RF_{zk} and $\text{RF}_{\text{a-coin}}$ are functionality maintaining. The transparency property of the firewall follows from the transparency of RF_{zk} and $\text{RF}_{\text{a-coin}}$. If RF_{zk} or $\text{RF}_{\text{a-coin}}$ detects failure then RF_{mpc} also detects a failure.

Next, we argue weak exfiltration resistance provided by RF_{mpc} . In Π_{mpc} the protocol messages of $\Pi_{\text{sh-mpc}}$ are deterministic given the input and randomness of each party. We know that the parties are weakly tampered, i.e. the tampering maintains functionality and does not abort. The randomness is generated from $\Pi_{\text{a-coin}}$ and the protocol enforces the party to use the same randomness from $\Pi_{\text{a-coin}}$ to compute the input commitments and the $\Pi_{\text{sh-mpc}}$ protocol steps by giving ZK proofs. Thus, a tampered party can leak only in $\Pi_{\text{a-coin}}$ or through the ZK proofs. This leakage is prevented by the firewalls of $\Pi_{\text{a-coin}}$ and Π_{zk} . We prove this claim through a sequence of hybrids. Let H and C denote the set of honest and corrupt parties respectively.

- Hyb_0 : This is the execution of Π_{mpc} between the set of tampered honest parties $\{\overline{P}_i\}_{i \in H}$ and corrupt parties $\{P_j\}_{j \in C}$.
 - Hyb_1 : This is the execution of Π_{mpc} between the set of tampered honest parties $\{\overline{P}_i\}_{i \in H}$ and corrupt parties $\{P_j\}_{j \in C}$ where the tampered parties run the honest implementation of Π_{zk} using true randomness and rest of the computation is performed according to Hyb_0 .
- The two hybrids are indistinguishable due to weak exfiltration resistance property provided by RF_{zk} .
- Hyb_2 : This is the execution of Π_{mpc} between the set of tampered honest parties $\{P_i\}_{i \in H}$ and corrupt parties $\{P_j\}_{j \in C}$ where every party P_i runs their honest implementation.

The two hybrids are indistinguishable due to weak exfiltration resistance property provided by $\text{RF}_{\text{a-coin}}$.

□

We have shown in Thm. 12 that Π_{mpc} is adaptively secure. Now, consider $\Pi_{\text{mpc}}^{\text{RF}}$, a firewalled version of the protocol Π_{mpc} where all honest parties P_i have their respective firewalls RF_i

attached to them. From Thm. 13, and our implication from Thm. 5, we conclude weak security preservation.

Theorem 14. $\Pi_{\text{mpc}}^{\text{RF}}$ is an actively secure MPC protocol against adaptively corruption of parties, and in the presence of functionality maintaining tampering of honest parties.

Instantiation. We obtain Π_{zk} and $\Pi_{\text{a-coin}}$ based on LWE and DDH assumptions respectively. We assume that Π_{zk} setup consists of n setup strings $-\{\text{urs}_{\text{zk}}^i\}_{i \in [n]}$, where urs_{zk}^i is used by party P_i to prove statements. Similarly, $\Pi_{\text{a-coin}}$ consists of n setup strings $-\{\text{urs}_{\text{a-coin}}^i\}_{i \in [n]}$, where $\text{urs}_{\text{a-coin}}^i$ is used in a session where party P_i is the initiating party. The commitment scheme can be instantiated from the adaptively secure commitment of [CSW20a] based on DDH in the urs_{com} model. We assume that Com consists of n setup strings $-\{\text{urs}_{\text{com}}^i\}_{i \in [n]}$, where $\text{urs}_{\text{com}}^i$ is used by party P_i to commit. We obtain an adaptively secure semi-honest MPC protocol in the urs model as follows. The work of [BLPV18] obtain a two-round semi-honest adaptively secure MPC protocol based on adaptively secure two-round OT protocol and augmented NCE. The work of [CSW20a] construct an adaptively secure two-round OT protocol from DDH in urs model and instantiate [CDMW09] the augmented NCE from DDH; thus yielding a two-round adaptively secure MPC protocol in the urs model from DDH. We instantiate $\Pi_{\text{sh-mpc}}$ using the DDH-based MPC protocol of [CSW20a].

Round complexity. In Π_{mpc} the subprotocols $\Pi_{\text{a-coin}}$ and the first two rounds of Π_{zk} can be run in parallel during the offline phase. Thus, the offline phase requires 3 rounds in total. The input commitment phase requires 1 round and $\Pi_{\text{sh-mpc}}$ requires 2 rounds when instantiated using the protocol of [CSW20a]. We get a 6 round MPC protocol from DDH and LWE. Our result is outlined in Thm. 15.

Theorem 15. Assuming DDH and LWE assumption holds, then Π_{mpc} is an actively secure MPC protocol computing functionality \mathcal{F}_{mpc} under reverse firewalls against adaptive corruption of parties and runs within 6 rounds in urs_{mpc} model.

8 Adaptively-Secure Multi-party Coin-Tossing Protocol with Reverse Firewalls in the Plain Model

In this section we give a protocol in the plain model with reverse firewalls to generate the setup string urs_{mpc} required for Π_{mpc} . A high level overview of our construction can be found in Sec. 1.2 and our protocol is presented in Fig. 21. Our protocol satisfies security against adaptive corruptions in the plain model and its security is summarized in Thm. 16.

Theorem 16. Let Discrete Log and Knowledge of Exponent Assumptions hold in a bilinear group \mathbb{G} (see Sec. 2.1) and PKE is a public key encryption with oblivious ciphertext sampling, oblivious public key sampling, satisfying additive homomorphism over key space, message space, randomness space and ciphertext space with public key space being \mathbb{Z}_q . Π_{coin} (Fig. 21) securely implements coin-tossing functionality (Def. 13) against adaptive corruptions in the plain model.

Proof. We prove that Π_{coin} is adaptively secure by providing an adaptive simulator Sim_{coin} in Fig. 23, 24. We prove indistinguishability between the real and ideal world adversary view through a sequence of hybrids:

- Hyb_0 : This is the real world execution of the protocol.

Fig. 21. Adaptively-Secure Multi-party Coin-Tossing Protocol Π_{coin} using Reverse Firewalls without Setup

– **Public Inputs:** Each party gets as input the group \mathbb{G} where DLP assumption holds. Every element $g' \in \mathbb{G} \setminus 1$ is a generator. Every party also receives a generator g as input and a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{H}$. Let $\text{PKE} = (\text{Gen}, \text{Enc}, \text{Dec}, \text{oGen}, \text{oEnc})$ is a public key encryption with oblivious ciphertext sampling and oblivious public keys sampling satisfying additive homomorphism over key space, message space, randomness space and ciphertext space. Let the public key space of PKE be \mathbb{Z}_q .

Parameter Generation Phase:

The following protocol steps are repeated by every party P_i for $i \in [n]$ with every party P_j for $j \in [n] \setminus i$ to generate the pair-wise commitment parameter h_{ij} where P_i is the committer and P_j is the verifier. We denote it as h and the pairwise communication without any subscript ij to avoid notation overloading.

1. P_j samples an $R \leftarrow \mathbb{G}$. P_j sends R to P_i .
2. P_i aborts if $R \in \{1, g\}$. P_i samples $a_1, u \leftarrow \mathbb{Z}_q$ and computes $A_1 = g^{a_1}$ and commits to a_1 using randomness u as $v = A_1 R^u$. P_i also samples $\text{pk}_1 \leftarrow \text{oGen}(1^\lambda)$ and commits to it as $v_p = g^{\text{pk}_1} R^{u_p}$ for $u_p \leftarrow \mathbb{Z}_q$. P_i sends (v, v_p) to P_j .
3. P_j samples $A_2 \leftarrow \mathbb{G}$, $\text{pk}_2 \leftarrow \text{oGen}(1^\lambda)$, and sends (A_2, pk_2) to P_i .
4. P_i opens commitment v_p by sending (u_p, pk_1) . P_i also sends u to P_j . P_i computes $\text{pk} = \text{pk}_1 + \text{pk}_2$.
5. P_j aborts if (u_p, pk_1) is not a valid opening of v_p . Else, P_j computes $\text{pk} = \text{pk}_1 + \text{pk}_2$. P_j computes $A_1 = \frac{v}{R^u}$ and sets $A = A_1 \cdot A_2$. P_j samples commitment trapdoor $t \leftarrow \mathbb{Z}_q$ and sets commitment parameter as $h = g^t$. P_j sends (h, A^t) to P_i .
6. P_i computes $A = A_1 \cdot A_2$ and aborts if $e(h, A) \neq e(g, Z)$ where P_i received (h, Z) from P_j . Else, P_i proves knowledge of discrete log of A_1 by sending a_1 to P_j . If $v \neq g^{a_1} R^u$ then P_j aborts. Else, set (g, h) as the pair-wise commitment parameter and pk as the pairwise encryption parameter.

Commitment Generation Phase:

Every party P_i chooses a random coin $s_i \leftarrow \{0, 1\}$, and commits to it pairwise. For $i \in [n]$, every party P_i and constructs $c_{ij} = g^{s_i} h_{ij}^{d_{ij}}$ where h_{ij} is the pairwise commitment parameter. P_i also encrypts the commitment randomness as $e_{ij, s_i} = \text{Enc}(\text{pk}_{ij}, d_{ij}; y_{ij})$ using randomness y_{ij} and samples $e_{ij, \bar{s}_i} \leftarrow \text{oEnc}(\text{pk}_{ij})$, where pk_{ij} is the pair-wise encryption parameter. P_i sends $(c_{ij}, e_{ij, 0}, e_{ij, 1})$ to P_j .

Commitment Opening Phase:

For all $i \in [n]$, party P_i broadcasts s_i . P_i opens c_{ij} pairwise (for all $j \in [n] \setminus i$) by sending (d_{ij}, y_{ij}) and claims that e_{ij, \bar{s}_i} was obliviously sampled.

Output Phase:

For all $i \in [n]$, party P_i verifies the commitments $c_{ji} \stackrel{?}{=} g^{s_j} h_{ji}^{d_{ji}}$ and $e_{ji, s_j} = \text{Enc}(\text{pk}_{ji}, d_{ji}; y_{ji})$. If all verification checks pass then P_i sets $S = (\sum_{k \in [n]} s_k) \bmod 2$ and outputs S as the final random coin.

Fig. 22. Reverse Firewall RF_i for Party P_i in Π_{coin}

Parameter Generation Phase:

The following protocol steps are repeated for every party P_j for $j \in [n] \setminus i$ where P_i is the committer and P_j is the verifier:

1. When P_j sends R , RF_i samples an $r \leftarrow \mathbb{Z}_q$ and sends $\hat{R} = R^r$ to P_i .
2. When P_i sends $v = A_1 \hat{R}^u$, RF_i forwards $\hat{v} = \hat{A}_1 R^{\hat{u}}$ to P_j where $\hat{A}_1 = A_1 \cdot \hat{A}$, $\hat{v} = v \cdot \hat{A} \cdot R^{\hat{u}}$ and $\hat{u} = ru + \tilde{u}$ for random values $\tilde{u} \leftarrow \mathbb{Z}_q$ and $\hat{A} = g^{\tilde{a}}$. When P_i sends $v_p = g^{\text{pk}_1} \hat{R}^{u_p}$, RF_i forwards $\hat{v}_p = g^{\text{pk}_1} R^{\hat{u}_p}$ to P_j where $\hat{\text{pk}}_1 = \text{pk}_1 + \tilde{\text{pk}}$, $\hat{v} = v \cdot g^{\tilde{\text{pk}}} \cdot R^{\hat{u}_p}$ and $\hat{u} = ru_p + \tilde{u}_p$ for random values $\tilde{u}_p \leftarrow \mathbb{Z}_q$ and $\tilde{\text{pk}} = \text{oGen}(1^\lambda)$.
3. When P_j sends (A_2, pk_2) , RF_i forwards $\hat{A}_2 = A_2 \cdot \hat{A}$ and $\hat{\text{pk}}_2 = \text{pk}_2 + \tilde{\text{pk}}$ to P_i . P_i computes $\hat{\text{pk}} = \text{pk}_1 + \hat{\text{pk}}_2 = \text{pk}_1 + \text{pk}_2 + \tilde{\text{pk}}$.
4. When P_i sends commitment randomness (u, u_p, pk_1) , RF_i drops the message if $v_p \neq g^{u_p} \hat{R}^{\text{pk}_1}$. Else, RF_i forwards $(\tilde{u}, \tilde{u}_p, \tilde{\text{pk}}_1)$ to P_j .
5. P_j computes $\hat{\text{pk}} = \hat{\text{pk}}_1 + \hat{\text{pk}}_2 = \text{pk}_1 + \text{pk}_2 + \tilde{\text{pk}}$. P_j computes $\hat{A} = A_1 \cdot A_2 \cdot \hat{A}$. When P_j sends $(h, Z) = (h, \hat{A}^t)$ drop the message if $e(h, \hat{A}) \neq e(g, Z)$. Else, sample a $\tilde{t} \leftarrow \mathbb{Z}_q$ and compute $\hat{h} = h^{\tilde{t}}$. Send $(\hat{h}, \hat{Z}) = (\hat{h}, Z^{\tilde{t}})$ to P_i . P_j sets h as the parameter and P_i sets $\hat{h} = h^{\tilde{t}}$ as the parameter. P_i and P_j sets $\hat{\text{pk}}$ as the pairwise public key parameter.
6. When P_i sends a_1 , RF_i drops the message if $v \neq g^{a_1} \hat{R}^u$. Else, RF_i forwards $\hat{a}_1 = a_1 + \tilde{a}$ to P_j .

The above steps are also repeated when P_i is the verifier and P_j is the committer.

Commitment Generation Phase:

RF_i chooses a random coin \tilde{s}_i and computes $\{s_{ji}\}_{j \in [n] \setminus i}$ randomly such that $\sum_{j \in [n] \setminus i} s_{ji} = \tilde{s}_i$. RF_i performs the following :

- P_i is the committer: When P_i sends a commitment $(c_{ij}, e_{ij,0}, e_{ij,1})$ compute $\hat{c}_{ij} = g^{\hat{s}_i} h_{ij}^{\hat{d}_{ij}} = c_{ij} \cdot g^{\tilde{s}_i} \cdot h_{ij}^{\tilde{d}_{ij}}$ where $\hat{s}_i = s_i + \tilde{s}_i$ and $\hat{d}_{ij} = d_{ij} \cdot \tilde{t} + \tilde{d}_{ij}$ for $\tilde{d}_{ij} \leftarrow \mathbb{Z}_q$. Set $e_{ij,0} = \tilde{t} \cdot e_{ij,0} + \text{Enc}(\hat{\text{pk}}, \tilde{d}_{ij}; y_{ij,0})$ and $e_{ij,1} = \tilde{t} \cdot e_{ij,1} + \text{Enc}(\hat{\text{pk}}, \tilde{d}_{ij}; y_{ij,1})$. The firewall forwards $(\hat{c}_{ij}, e_{ij,0}, e_{ij,1})$ to P_j . Here \tilde{t}, h_{ij} and $\hat{\text{pk}}$ correspond to the run where P_j is verifier and P_i is committer.
- P_j is the committer: When P_j sends a commitment $(c_{ji}, e_{ji,0}, e_{ji,1})$ compute $\hat{c}_{ji} = g^{\hat{s}_j} h_{ji}^{\hat{d}_{ji}} = c_{ji} \cdot g^{\tilde{s}_j} \cdot h_{ji}^{\tilde{d}_{ji}}$ where $\hat{s}_j = s_j + \tilde{s}_j$ and $\hat{d}_{ji} = d_{ji} \cdot \tilde{t} + \tilde{d}_{ji}$ for $\tilde{d}_{ji} \leftarrow \mathbb{Z}_q$. Set $e_{ji,0} = \tilde{t} \cdot e_{ji,0} + \text{Enc}(\hat{\text{pk}}, \tilde{d}_{ji}; y_{ji,0})$ and $e_{ji,1} = \tilde{t} \cdot e_{ji,1} + \text{Enc}(\hat{\text{pk}}, \tilde{d}_{ji}; y_{ji,1})$. The firewall forwards $(\hat{c}_{ji}, e_{ji,0}, e_{ji,1})$ to P_i . Here \tilde{t}, h_{ji} and $\hat{\text{pk}}$ correspond to the run where P_i is verifier and P_j is committer.

Commitment Opening Phase:

- When party P_i broadcasts s_i , RF_i broadcasts \hat{s}_i . When P_i opens commitments by sending (d_{ij}, y_{ij}) , RF_i drops the message if $c_{ij} \neq g^{\hat{s}_i} h_{ij}^{\hat{d}_{ij}}$ or $e_{ij,s_i} \neq \text{Enc}(\hat{\text{pk}}, d_{ij}; y_{ij})$. Else, RF_i sends $(\hat{d}_{ij}, \hat{y}_{ij}) = (\tilde{t} \cdot d_{ij} + \tilde{d}_{ij}, \tilde{t} \cdot y_{ij} + \tilde{y}_{ij})$ and claims that $e_{ij,1-\hat{s}_i}$ was obviously sampled.
- When party P_j broadcasts s_j , RF_i sends \hat{s}_j to P_i . When P_j opens commitments by sending (d_{ji}, y_{ji}) , RF_i drops the message if $c_{ji} \neq g^{\hat{s}_j} h_{ji}^{\hat{d}_{ji}}$ or $e_{ji,s_j} \neq \text{Enc}(\hat{\text{pk}}, d_{ji}; y_{ji})$. Else, RF_i sends $(\hat{d}_{ji}, \hat{y}_{ji}) = (\tilde{t} \cdot d_{ji} + \tilde{d}_{ji}, \tilde{t} \cdot y_{ji} + \tilde{y}_{ji})$ and claims that $e_{ji,1-\hat{s}_j}$ was obviously sampled.

- **Hyb₁** : Same as **Hyb₀**, except Sim_{coin} aborts if P_j^* is corrupt and $\mathcal{X}_{\mathcal{A}}$ fails to extract a valid trapdoor t in the parameter generation phase.

When P_j^* is corrupt in parameter generation phase and P_i is honest, A is randomly generated since A_1 is random and v perfectly hides A_1 . Indistinguishability follows due to the existence of a knowledge extractor $\mathcal{X}_{\mathcal{A}}$ for adversary \mathcal{A} .

- **Hyb₂** : Same as **Hyb₁**, except Sim_{coin} rewinds to step 3 (after running parameter generation phase till step 6) when P_i^* is corrupt and P_j is honest and Sim_{coin} aborts if P_i^* sends $(a'_1, u') \neq (a, u)$ or $(u'_p, \text{pk}'_1) \neq (u_p, \text{pk}_1)$.

Indistinguishability follows due to the binding property of the Pedersen Commitment scheme based on Discrete Log assumption. If both (a_1, u) and (a'_1, u') are valid openings of v then the discrete log of R can be computed, breaking DL assumption. The reduction obtains a random challenge R from the DL challenger and forwards it to P_j^* in step 1. After obtaining two different valid openings (a_1, u) and (a'_1, u') , the reduction outputs the discrete log of R as $\frac{a'_1 - a_1}{u_1 - u'_1}$. The same argument holds if both (u'_p, pk'_1) and (u_p, pk_1) are valid openings.

- **Hyb₃** : Same as **Hyb₂**, except Sim_{coin} samples $(\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^\lambda)$ and sets $\text{pk}'_2 = \text{pk} - \text{pk}_1$ after rewinding.

In **Hyb₂**, the public key pk is a random public key and in **Hyb₃** the public key pk is sampled using Gen algorithm. An adversary who can distinguish between the two can distinguish a random public key from an honest public key breaking oblivious public key sampling property. When we instantiate our encryption scheme using the [GSW13] this indistinguishability follows from the LWE assumption since the honest public key is an LWE sample. The reduction can plug in the challenge public key pk' by setting $\text{pk}'_2 = \text{pk}' - \text{pk}_1$.

- **Hyb₄** : Same as **Hyb₃**, except Sim_{coin} computes $(c_{ij}, e_{ij,0}, e_{ij,1})$ in the equivocal mode (using pairwise commitment trapdoor t) for an honest P_i such that c_{ij} opens to both 0 and 1.

The commitment c_{ij} perfectly hides the committed bit s_i . In **Hyb₃**, $e_{ij, \overline{s_i}}$ is obviously sampled whereas in **Hyb₄** it is a valid encryption. Indistinguishability follows from the oblivious ciphertext sampleability.

- **Hyb₅** : Same as **Hyb₄**, except Sim_{coin} extracts $d_{ji,0} = \text{Dec}(\text{sk}, e_{ji,0})$ and $d_{ji,1} = \text{Dec}(\text{sk}, e_{ji,1})$ and aborts in the parameter generation phase if $c_{ji} = h^{d_{ji,0}} = gh^{d_{ji,1}}$ for a corrupt P_j^* .

Indistinguishability follows from the Discrete Log assumption. If P_i^* finds two valid openings $(0, d_{ij,0})$ and $(1, d_{ij,1})$ then P_i^* finds discrete log of h_{ij} as $\frac{1}{d_{ij,0} - d_{ij,1}}$. Consider the parameter generation phase where P_i^* is the corrupt committer and P_j is the simulated verifier. The reduction obtains A_1 and after the first rewinding it samples an $A' = g^{a'}$ and sets $A'_2 = \frac{A'}{A_1}$. The simulated P_j sets $h_{ij} = X$ as the commitment parameter, where X is the discrete log challenge. If P_i^* breaks binding of c_{ij} then it finds discrete log of X as explained above.

- **Hyb₆** : Same as **Hyb₅**, except Sim_{coin} invokes the coin tossing functionality to obtain the simulated coin s . Sim_{coin} considers an honest party P_k and changes its random coin to s_k (by following the simulation algorithm) and opens c_{kj} as $(d_{kj, s_k}, y_{kj, s_k})$ such that the output coin equals to the simulated coin. Sim_{coin} also claims that $e_{kj, \overline{s_k}}$ was obviously sampled. Sim_{coin} also equivocates the commitments c_{ij} to s_i . This is the ideal world execution of the protocol.

Indistinguishability follows due to the equivocal property of the Pedersen commitment scheme and the oblivious ciphertext sampleability of the PKE. Sim_{coin} has successfully extracted the trapdoors t_{kj} for all parties P_j . Using these trapdoors it can equivocate the commitments c_{kj} correctly.

Non-malleability. Non-malleability among different parameter generation subprotocols is ensured as follows:

- P_i is honest and P_j^* is corrupt A corrupt P_j^* can try to forward h from a simulated session (where the committer is corrupt and verifier is honest) in this session. However, P_i is honest and generates a random A_1 and as a result A is random. If a corrupt P_j^* forwards h from a different simulated subsession without knowing the trapdoor then P_j^* fails in the check $e(h, A) \stackrel{?}{=} e(g, A^t)$.
- P_i^* is corrupt and P_j is honest In this case the parameter h is going to be randomly generated by an honest P_j . A corrupt P_i^* cannot forward a simulated commitment (where the committer is honest and verifier is corrupt) under an $h' \neq h$ (where h' was maliciously generated) since P_i^* needs to compute discrete log of h to produce a valid opening of the commitment under h given a valid opening under h' .

This completes our proof. The probability of abort in the real and ideal world is identical since in both worlds the adversary sees only the final output coin in the commitment opening phase. This final coin is either the real world coin or the simulated coin. □

Next, we turn to constructing a reverse firewall for Π_{coin} . We provide a reverse firewall RF_i for the tampered honest party P_i in Fig. 22. Weak ER for P_i is summarized in Thm. 17.

Theorem 17. *Let RF_i be the reverse firewall for party P_i in Π_{coin} . If Discrete Log assumption and Knowledge of Exponent assumption holds in a bilinear group \mathbb{G} and PKE is a public key encryption with oblivious ciphertext sampling and oblivious public keys sampling satisfying additive homomorphism over key space, message space, randomness space and ciphertext space and public key space of PKE be \mathbb{Z}_q . Then RF_i is transparent, functionality maintaining and provides weak exfiltration resistance for party P_i against every other party $\{P_j\}_{j \in [n] \setminus i}$ with valid transcripts, and detects failure for P_i .*

Proof.

Next, we show that RF_i (Fig. 22) in Π_{coin} provides weak ER for P_i by proving Thm. 17. It can be verified that RF_i is functionality maintaining and transparent. We assume that every element in \mathbb{G} is a generator. This holds when $\mathbb{G} = QR_p$ where $p = 2q + 1$ such that p, q are both primes. A tampered/malicious verifier can generate a leaky pairwise commitment parameter h . The firewall re-randomizes this h to $\hat{h} = h^t$ where $t \leftarrow \mathbb{Z}_q$ and sends \hat{h} to the committer as the commitment parameter. Upon receiving a commitment $c = g^s \hat{h}^r$ from the committer to coin s under \hat{h} the firewall rerandomizes it to a commitment \hat{c} under h to coin $\hat{s} = s + \tilde{s}$. It is performed as $\hat{c} = g^{\hat{s}} h^{\hat{r}} = (c) \cdot g^{\tilde{s}} \cdot h^{\tilde{r}}$ where $\hat{r} = rt + \tilde{r}$ and $\hat{s} = s + \tilde{s}$. By rerandomizing the commitments the firewall also rerandomizes each party's coin and thus the final output coin is guaranteed to be random. A malicious verifier can also construct h in such a way that it doesn't pass the check for proof of trapdoor. The firewall can verify the check by computing the pairing equation. In case the check doesn't pass the firewall drops the message else it rerandomizes h to \hat{h} and rerandomizes the check accordingly. Without pairing this would not have been possible. The keys and the ciphertext for the encryption scheme can be rerandomized due to the homomorphic property.

We argue indistinguishability between the transcripts for $\text{RF}_i \circ P_i$ and $\text{RF}_i \circ \overline{P}_i$ for every step as follows:

- *Parameter Generation Phase:* Here P_j can send maliciously generated messages. RF_i verifies if they are from the correct distribution and follow the protocol steps. If they fail to do so, the firewall drops those messages and detects failures. Else, it sanitizes them and sends them to P_i .

Fig. 23. Adaptive Simulator for Π_{coin}

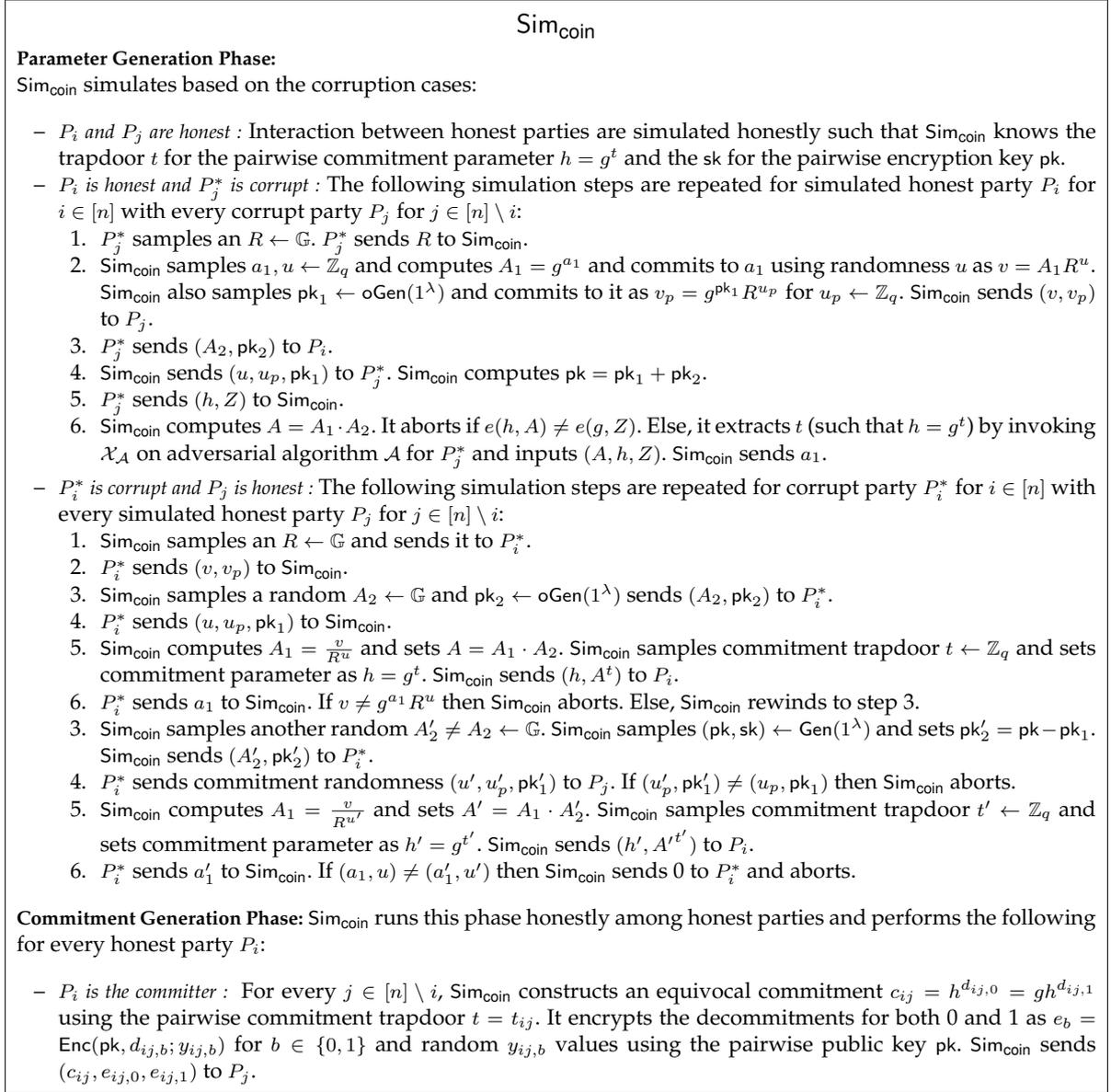


Fig. 24. Adaptive Simulator for Π_{coin} (Cont.)

Sim_{coin}

- P_i is the verifier : For every $j \in [n] \setminus i$, Sim_{coin} receives $(c_{ji}, e_{ji,0}, e_{ji,1})$. Sim_{coin} decrypts $d_{ji,b} = \text{Dec}(\text{sk}, e_{ji,b})$ using pairwise secret key $\text{sk} = \text{sk}_{ji}$ for $b \in \{0, 1\}$. Sim_{coin} aborts if $c_{ji} = h^{d_{ji,0}} = gh^{d_{ji,1}}$, i.e. P_j has broken the binding property of the commitment. Else, the simulator extracts the bit by setting $s_j = 0$ if $c_{ji} = h^{d_{ji,0}}$ or $s_j = 1$ if $c_{ji} = gh^{d_{ji,1}}$. Otherwise, Sim_{coin} sets $s_j = \perp$.

Commitment Opening Phase : For honest parties P_i ($i \neq k$), Sim_{coin} samples $s_i \leftarrow \{0, 1\}$ and broadcasts s_i . Sim_{coin} opens c_{ij} to s_i by sending (d_{ij,s_i}, y_{ij,s_i}) to P_j and claims that e_{ij,\bar{s}_i} was obliviously sampled. Sim_{coin} chooses an honest party P_k . Sim_{coin} invokes the coin tossing functionality to obtain output coin \mathbf{s} and computes $\mathbf{s}' = (\sum_{\ell \in [n] \setminus k} s_\ell) \bmod 2$. Sim_{coin} sets $s_k = \mathbf{s} - \mathbf{s}'$. Sim_{coin} broadcasts s_k and opens c_{kj} to s_k by sending (d_{kj,s_k}, y_{kj,s_k}) to P_j and claims that e_{kj,\bar{s}_k} was obliviously sampled.

Output Phase:
Sim_{coin} verifies all the decommitments sent by the corrupt parties and aborts if any decommitment is invalid. Else, Sim_{coin} outputs \mathbf{s} as the final random coin.

Simulating Adaptive Corruption of Parties:
When an honest party P_i gets corrupted the simulator can forward the view of the party to the adversary corresponding to the honest view. The knowledge of the pairwise commitment trapdoors and the pairwise secret keys are not provided to the adversary.

1. Here $\hat{R} = R^r$ is random since r is randomly sampled.
 2. When P_i sends v , RF_i forwards \hat{v} . A_1 is rerandomized to $\hat{A}_1 = A_1 \cdot \tilde{A}$ and \hat{v} is a rerandomized commitment under h where the randomness is \hat{u} . It is indistinguishable from an honestly generated commitment due to the randomness of \tilde{A} and \tilde{u} . When P_i sends v_u , RF_i forwards \hat{v}_u . pk_1 is rerandomized to $\hat{\text{pk}}_1 = \text{pk}_1 + \tilde{\text{pk}}$ and \hat{v}_u is a rerandomized commitment to $\hat{\text{pk}}_1$ under fresh randomness \hat{u}_p . $\hat{\text{pk}}_1$ is indistinguishable from a random public key due to the oblivious public key sampleability property.
 3. When P_j sends (A_2, pk_2) , RF_i forwards $\hat{A}_2 = A_2 \cdot \tilde{A}_1$ and $\hat{\text{pk}}_2 = \text{pk}_2 + \tilde{\text{pk}}$ to P_i . \hat{A}_2 looks random to P_i due to \tilde{A} . $\hat{\text{pk}}_2$ is indistinguishable from a random public key due to the oblivious public key sampleability property.
 4. When P_i sends commitment randomness (u, u_p, pk_1) , RF_i drops the message if v_u does not decommit correctly with (u_p, pk_1) . Else, it forwards $(\hat{u}, \hat{u}_p, \hat{\text{pk}}_1)$ to P_j . It is indistinguishable from an honestly generated one due to the randomness of \tilde{u} , \tilde{u}_p , and $\tilde{\text{pk}}$.
 5. Here the firewall drops the message and detects failure if the check does not pass. Else, it samples a random $\tilde{b} \leftarrow \mathbb{Z}_q$ and computes $(\hat{h}, \hat{Z}) = (\hat{h}, \tilde{Z}^{\tilde{b}})$ and sends it to P_i . \hat{h} is randomly distributed due to \tilde{b} .
 6. When P_i sends a_1 , RF_i drops the message if $v \neq g^{a_1} \hat{R}^u$. Else, RF_i forwards $\hat{a}_1 = a_1 + \tilde{a}$ to P_j . This step is indistinguishable due to \tilde{a}_1 .
- **Commitment Generation Phase:** RF_i chooses the random coin \tilde{s}_i such that the final coin outputted by the parties will be $\hat{s} = \tilde{s}_i + \sum_{k \in [n]} s_k$ which is random due to \tilde{s}_i . RF_i also computes \tilde{s}_{ji} such that $\sum_{j \in [n] \setminus i} \tilde{s}_{ji} = \tilde{s}_i$.
 - When P_i sends a commitment to s_i as $(c_{ij}, e_{ij,0}, e_{ij,1})$ the firewall rerandomizes it to $(\hat{c}_{ij}, e_{ij,0}, e_{ij,1})$ such that \hat{c}_{ij} is a rerandomized commitment to $\hat{s}_i = s_i + \tilde{s}_i$ under fresh randomness \hat{d}_{ij} . The encryptions are rerandomized and the corresponding decommitments are encrypted. This can be performed since the encryption scheme and the commitment scheme is additively homomorphic over the message and randomness space.
 - When P_j sends a commitment to s_j as $(c_{ji}, e_{ji,0}, e_{ji,1})$ the firewall rerandomizes it to $(\hat{c}_{ji}, e_{ji,0}, e_{ji,1})$ such that \hat{c}_{ji} is a rerandomized commitment to $\hat{s}_j = s_j + \tilde{s}_{ji}$ under fresh randomness \hat{d}_{ji} . The encryptions are rerandomized and the corresponding decommit-

ments are encrypted. This can be performed since the encryption scheme and the commitment scheme is additively homomorphic over the message and randomness space.

- *Commitment Opening Phase*: RF_i provides consistent openings using its internal state from previous step. The final coin outputted by the parties is $\hat{s} = \tilde{s}_i + \sum_k s_k$ which is random due to \tilde{s}_i .

We can also construct a Transform function for RF_i and P_i which returns a composed state for $\text{RF}_i \circ P_i$ when P_i adaptively corrupted.

- In the parameter generation phase, when P_i is the committer the Transform function takes in input $(u, u_p, \text{pk}_1, a_1, \hat{R}, A_1, A_2, A, v, v_u, \hat{Z}, \hat{h}, \text{pk}_2)$ from P_i and $(\tilde{u}, \tilde{u}_p, \text{pk}_1, \tilde{a}_1, \tilde{A})$ from RF_i and returns $(\hat{u}, \hat{u}_p, \hat{\text{pk}}_1, \hat{a}_1, R, \hat{A}_1, A_2, \hat{A}, \hat{v}, \hat{v}_u, h, Z, \text{pk}_2)$.
- In the parameter generation phase, when P_i is the verifier the Transform function takes in $(t, R, \hat{A}_1, A_2, \hat{A}, \hat{v}, Z, h, \hat{v}_p, \hat{u}_p, \text{pk}_1)$ from P_i and (\tilde{r}, \tilde{t}) from RF_i and returns $(\hat{t}, \hat{R}, A_1, \hat{A}_2, \hat{A}, v, \hat{Z}, \hat{h}, v_p, u_p, \text{pk}_1)$ where $\hat{t} = t \cdot \tilde{t}$.
- In the commitment generation phase, when P_i is the committer the Transform function takes in input $(s_i, c_{ij}, d_{ij}, e_{ij,0}, e_{ij,1}, y_{ij})$ from P_i and $(\tilde{s}_i, \tilde{d}_{ij}, y_{ij,0}, y_{ij,1}, \hat{\text{pk}}, \tilde{t})$ from RF_i , and returns $(\hat{s}_i, \hat{c}_{ij}, \hat{d}_{ij}, e_{ij,0}, e_{ij,1}, \hat{y}_{ij})$.
- In the commitment generation phase, when P_i is the verifier the Transform function takes in input $(c_{ji}, e_{ji,0}, e_{ji,1})$ from P_i and $(\tilde{s}_{ji}, \tilde{d}_{ji}, y_{ji,0}, y_{ji,1}, \hat{\text{pk}}, \tilde{t})$ from RF_i , and returns $(\hat{c}_{ji}, e_{ji,0}, e_{ji,1})$.
- In the commitment opening phase, when P_i is the committer RF_i takes in input (s_i, d_{ij}, y_{ij}) from P_i and $(\tilde{s}_i, \tilde{d}_{ij}, y_{ij,0}, y_{ij,1})$ from RF_i and returns $(\hat{s}_i, \hat{d}_{ij}, \hat{y}_{ij})$.

We have shown the final coin when there is only a firewall RF_i for one honest party P_i . The final protocol in the firewall setting will consist of firewall RF_j for every honest party $\{P_j\}_{j \in \mathcal{H}}$. In such a case, the final coin will be $s + \sum_{j \in \mathcal{H}} \tilde{s}_j$. \square

In the final protocol every honest party $\{P_j\}_{j \in \mathcal{H}}$ will have a firewall $\{\text{RF}_j\}_{j \in \mathcal{H}}$ and the firewalls will be composed together. By applying the result of Thm. 5 we obtain the following result.

Theorem 18. *If Discrete Log and Knowledge of Exponent assumptions hold in a bilinear group \mathbb{G} and PKE is a public key encryption with oblivious ciphertext sampling and oblivious public keys sampling satisfying additive homomorphism over key space, message space, randomness space, ciphertext space and public key space of PKE be \mathbb{Z}_q . Then Π_{coin} (Fig. 21) securely implements the coin-tossing functionality (Def. 13) against adaptive corruption of parties in the plain model and in the presence of functionality maintaining tampering of honest parties.*

We observe that the fully homomorphic encryption scheme of [GSW13] based on LWE assumption satisfies all the properties required from the PKE. We consider $q = \max(q_{\text{LWE}}, q_{\text{DL}})$ where LWE holds for $q \geq q_{\text{LWE}}$ and DL holds for $q \geq q_{\text{DL}}$. Thus we get the following result:

Theorem 19. *If Discrete Log and Knowledge of Exponent assumption holds in a bilinear group \mathbb{G} and LWE assumption holds. Then Π_{coin} (Fig. 21) securely implements the coin-tossing functionality (Def. 13) against adaptive corruption of parties in the plain model and in the presence of functionality maintaining tampering of honest parties.*

9 The Final Compiler

In this section, we show our final result, i.e., an *adaptively* secure MPC protocol in the *plain* model that admits reverse firewalls. In particular, the reverse firewall for our final MPC protocol is obtained by combining the reverse firewall for our adaptively secure MPC protocol Π_{mpc} in the uniform random string (urs_{mpc}) model (see Section 7.3) along with the reverse firewall for our adaptively secure multi-party coin-tossing protocol Π_{coin} in the plain model (see Section 8). Let us denote the final MPC protocol (in the plain model) to be Π which is obtained by first running Π_{coin} to obtain urs_{mpc} and then running Π_{mpc} using urs_{mpc} .

Let us consider a reverse firewall $\text{RF}_i = (\text{RF}_{\text{coin}}^i, \text{RF}_{\text{mpc}}^i)$ to be the firewall for a party P_i in the protocol Π . RF_i is obtained by first applying $\text{RF}_{\text{coin}}^i$ to the messages of Π_{coin} , followed by application of RF_{mpc}^i to the messages of Π_{mpc} , if $\text{RF}_{\text{coin}}^i$ did not output \perp . We show that RF_i provides weak ER for party P_i in Π .

Theorem 20 (Composition Theorem for Π). *Let Π_{mpc} be an adaptively secure MPC protocol in the uniform random string (urs_{mpc}) model, Π_{coin} securely implement the coin-tossing functionality (see Def. 13) against adaptive corruption of parties in the plain model. Let RF_{mpc}^i and $\text{RF}_{\text{coin}}^i$ be transparent, functionality-maintaining, and weakly exfiltration-resistant reverse firewalls for some party P_i in Π_{mpc} and Π_{coin} respectively. Then RF_i is transparent, functionality-maintaining, and weakly exfiltration-resistant reverse firewall for party P_i in the protocol Π .*

Proof. The proof of this theorem is similar to the proof of Thm. 13 (see Sec. 7.3). In particular, Π_{coin} generates the URS urs_{mpc} and the protocol Π_{mpc} enforces all parties to use urs_{mpc} (since the parties are weakly tampered). Thus, a tampered party can leak only in Π_{coin} or through Π_{mpc} . However, this leakage is prevented by the firewalls $\text{RF}_{\text{coin}}^i$ and RF_{mpc}^i . We prove this claim through a sequence of hybrids. Let H and C denote the set of honest and corrupt parties respectively.

- Hyb_0 : This is the execution of Π between the set of tampered honest parties $\{\overline{P}_i\}_{i \in H}$ and corrupt parties $\{P_j\}_{j \in C}$.
- Hyb_1 : This is the execution of Π between the set of tampered honest parties $\{\overline{P}_i\}_{i \in H}$ and corrupt parties $\{P_j\}_{j \in C}$ where the tampered parties run the honest implementation of Π_{coin} using true randomness and rest of the computation is performed according to Hyb_0 . Since Π_{coin} implements an ideal coin-tossing functionality (see Def. 13) all the parties receive a random string urs_{mpc} at the end of the protocol execution. The two hybrids are indistinguishable due to *weak exfiltration resistance* property provided by RF_{coin} .
- Hyb_2 : This is the execution of Π between the set of tampered honest parties $\{P_i\}_{i \in H}$ and corrupt parties $\{P_j\}_{j \in C}$ where every party P_i runs their honest implementation using urs_{mpc} as the urs obtained from Π_{coin} . The two hybrids are indistinguishable due to *weak ER* property provided by RF_{mpc} .

□

References

- ABK18. Benedikt Auerbach, Mihir Bellare, and Eike Kiltz. Public-key encryption resistant to parameter subversion and its realization from efficiently-embeddable groups. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 348–377. Springer, Heidelberg, March 2018.
- AF07. Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 118–136. Springer, Heidelberg, February 2007.
- AMV15. Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 364–375. ACM Press, October 2015.
- BBG⁺13. James Ball, Julian Borger, Glenn Greenwald, et al. Revealed: how us and uk spy agencies defeat internet privacy and security. *Know Your Neighborhood*, 2013.

- BCJ21. Pascal Bemmam, Rongmao Chen, and Tibor Jager. Subversion-resilient public key encryption with practical watchdogs. In *PKC 2021, Virtual Event, May 10-13, 2021, Proceedings, Part I*, volume 12710 of *LNCS*, pages 627–658. Springer, 2021.
- BFS16. Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, December 2016.
- BJK15. Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1431–1440. ACM Press, October 2015.
- BLPV18. Fabrice Benhamouda, Huijia Lin, Antigoni Polychroniadou, and Muthuramakrishnan Venkatasubramanian. Two-round adaptively secure multiparty computation from standard assumptions. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 175–205. Springer, Heidelberg, November 2018.
- BPR14. Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 1–19. Springer, Heidelberg, August 2014.
- CDMW09. Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved non-committing encryption with applications to adaptively secure protocols. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 287–302. Springer, Heidelberg, December 2009.
- CDN20. Suvradip Chakraborty, Stefan Dziembowski, and Jesper Buus Nielsen. Reverse firewalls for actively secure MPCs. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 732–762. Springer, Heidelberg, August 2020.
- CHY20. Rongmao Chen, Xinyi Huang, and Moti Yung. Subvert KEM to break DEM: practical algorithm-substitution attacks on public-key encryption. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Proceedings, Part II*, volume 12492 of *LNCS*, pages 98–128. Springer, 2020.
- CMY⁺16. Rongmao Chen, Yi Mu, Guomin Yang, Willy Susilo, Fuchun Guo, and Mingwu Zhang. Cryptographic reverse firewall via malleable smooth projective hash functions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 844–876. Springer, Heidelberg, December 2016.
- CSW20a. Ran Canetti, Pratik Sarkar, and Xiao Wang. Efficient and round-optimal oblivious transfer and commitment with adaptive security. In *Advances in Cryptology - ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 277–308. Springer, 2020.
- CSW20b. Ran Canetti, Pratik Sarkar, and Xiao Wang. Triply adaptive uc nizk. Cryptology ePrint Archive, Report 2020/1212, 2020. <https://eprint.iacr.org/2020/1212>.
- DCM⁺19. Emma Dauterman, Henry Corrigan-Gibbs, David Mazières, Dan Boneh, and Dominic Rizzo. True2F: Backdoor-resistant authentication tokens. In *2019 IEEE Symposium on Security and Privacy*, pages 398–416. IEEE Computer Society Press, May 2019.
- DFP15. Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In *International Workshop on Fast Software Encryption*, pages 579–598. Springer, 2015.
- DGG⁺15. Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 101–126. Springer, Heidelberg, April 2015.
- DMS16. Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. Message transmission with reverse firewalls—secure communication on corrupted machines. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 341–372. Springer, Heidelberg, August 2016.
- DPSW16. Jean Paul Degabriele, Kenneth G. Paterson, Jacob C. N. Schuldt, and Joanne Woodage. Backdoors in pseudorandom number generators: Possibility and impossibility results. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 403–432. Springer, Heidelberg, August 2016.
- FLS99. Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.
- FM18. Marc Fischlin and Sogol Mazaheri. Self-guarding cryptographic protocols against algorithm substitution attacks. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 76–90. IEEE, 2018.
- GMV20. Chaya Ganesh, Bernardo Magri, and Daniele Venturi. Cryptographic reverse firewalls for interactive proof systems. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *ICALP 2020*, volume 168 of *LIPICs*, pages 55:1–55:16. Schloss Dagstuhl, July 2020.
- GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

- GS12. Sanjam Garg and Amit Sahai. Adaptively secure multi-party computation with dishonest majority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 105–123. Springer, Heidelberg, August 2012.
- GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.
- GVW15. Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015.
- MS15. Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 657–686. Springer, Heidelberg, April 2015.
- RTYZ16. Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 34–64. Springer, Heidelberg, December 2016.
- RTYZ17. Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 907–922. ACM Press, October / November 2017.
- SF07. Dan Shumow and Niels Ferguson. On the possibility of a back door in the nist sp800-90 dual ec prng. In *Proc. Crypto*, volume 7, 2007.
- Sim84. Gustavus J. Simmons. Authentication theory/coding theory. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 411–431. Springer, Heidelberg, August 1984.
- YY96. Adam Young and Moti Yung. The dark side of “black-box” cryptography, or: Should we trust capstone? In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 89–103. Springer, Heidelberg, August 1996.