Reverse Firewalls for Oblivious Transfer Extension and Applications to Zero-Knowledge

Suvradip Chakraborty[1], Chaya Ganesh[2], and Pratik Sarkar*[3]

[1] ETH Zurich
suvradip.chakraborty@inf.ethz.ch
[2] Indian Institute of Science, India
chaya@iisc.ac.in
[3] Department of Computer Science, Boston University
pratik93@bu.edu

Abstract. In the setting of subversion, an adversary tampers with the machines of the honest parties thus leaking the honest parties' secrets through the protocol transcript. The work of Mironov and Stephens-Davidowitz (*EUROCRYPT'15*) introduced the idea of reverse firewalls (RF) to protect against tampering of honest parties' machines. All known constructions in the RF framework rely on the malleability of the underlying operations in order for the RF to rerandomize/sanitize the transcript. RFs are thus limited to protocols that offer some structure, and hence based on public-key operations. In this work, we initiate the study of *efficient* Multiparty Computation (MPC) protocols in the presence of tampering. In this regard,

- We construct the *first* Oblivious Transfer (OT) extension protocol in the RF setting. We obtain $\mathsf{poly}(\kappa)$ maliciously-secure OTs using $\mathcal{O}(\kappa)$ public key operations and $\mathcal{O}(1)$ inexpensive symmetric key operations, where κ is the security parameter.
- We construct the *first* Zero-knowledge protocol in the RF setting where each multiplication gate can be proven using $\mathcal{O}(1)$ symmetric key operations. We achieve this using our OT extension protocol and by extending the ZK protocol of Quicksilver (Yang, Sarkar, Weng and Wang, *CCS'21*) to the RF setting.
- Along the way, we introduce new ideas for malleable interactive proofs that could be of independent interest. We define a notion of *full malleability* for Sigma protocols that unlike prior notions allow modifying the instance as well, in addition to the transcript. We construct new protocols that satisfy this notion, construct RFs for such protocols and use them in constructing our OT extension.

The key idea of our work is to demonstrate that correlated randomness may be obtained in an RF-friendly way *without* having to rerandomize the entire transcript. This enables us to avoid expensive public-key operations that grow with the circuit-size.

* Supported by NSF Awards 1931714, 1414119, and the DARPA SIEVE program.

Table of Contents

1 Introduction

Protocols in cryptography are proven secure under standard definitions where the assumption is that the honest parties trust their machines to implement their computation. This assumption breaks down in the real world, where even honest parties' computations are performed on untrusted machines. The security guarantees of these protocols fall short of protecting against attacks that take advantage of the *implementation* details instead of merely treating the algorithm as a black-box. Such attacks are indeed realistic, both because users are compelled to use third-party hardware due to lack of expertise, software mandated due to standardization, or even because of intentional *tampering* due to subversion. The threat of a powerful adversary modifying the implementation so that the subverted algorithm remains indistinguishable from the specification in black-box interface, while leaking secrets is not overkill. Snowden revelations [BBG+13] show that one of the potential mechanisms for large scale mass surveillance is subversion of cryptographic standards and tampering of hardware.

Reverse Firewalls. The framework of cryptographic reverse firewalls was introduced by Mironov and Stephens-Davidowitz [MS15] for designing protocols secure against adversaries that can corrupt the machines of honest parties in order to compromise their security. In such a setting, all parties are equipped with their own reverse firewall (RF), which sits between the party and the external world and sanitizes the parties' incoming and outgoing messages. The parties do not trust the RF, the RF cannot create security and the hope is for the RF to preserve security in the face of subversion. Roughly, the security properties desired from an RF are: (i) *exfiltration-resistance*: the firewall prevents the machine from leaking any information to the outside world regardless of how the user's machine behaves. (ii) *security preservation*: the protocol with the firewall is secure even when honest parties' machines are tampered.

The work of [MS15] provides a construction of a two-party passively secure computation protocol with a reverse firewall in addition to introducing the RF framework. Feasibility of RF for multi-party computation (MPC) was shown in [CDN20] who constructed RFs for MPC protocols in the malicious setting. The recent work of [CGPS21] constructs MPC protocols with RF in the presence of adaptive corruptions. We discuss other works in the RF framework and related models for subversion resistance in Section 1.3.

Motivation. We begin by observing that both existing works that construct RFs for maliciously-secure MPC protocols [CDN20, CGPS21] follow roughly the same template – that of the GMW compiler [GMW87]. Both constructions are essentially compilers: they take a semi-honest secure MPC protocol and run GMW-like steps in the reverse firewall setting to yield a secure MPC protocol with reverse firewalls. In the process, they design secure protocols for the underlying primitives (like augmented coin-tossing and zero knowledge) in the GMW compiler, construct reverse firewalls for each of the primitives, and finally, show that the compiled MPC protocol is secure in the presence of tampering of honest parties. This renders the resulting protocols inefficient for practical purposes.

The current techniques for constructing the RFs crucially make use of malleability. This is because, the constructions rely on the ability of the RF to randomize/maul messages to prevent exfiltration. In order to not break correctness, such mauling has to be on messages that are malleable and therefore requires the underlying primitives to be homomorphic. Indeed, the RFs for Sigma protocols of [GMV20] rely on malleability of Sigma protocol, and message and key homomorphism of Pedersen commitment. The RF of [CDN20] relies on controlled malleable non-interactive zero-knowledge proofs (NIZK), and the constructions of [CGPS21] need primitives like homomorphic commitment scheme and homomorphic public-key encryption. These randomization techniques for constructing the RF necessitates the MPC protocol to use homomorphic primitives based on expensive public-key operations. In particular, the GMW approach of [CDN20, CGPS21] require number of public-key operations that is proportional to the size of the circuit being computed by the protocol. However, progress in MPC has resulted in several efficient protocols [JKO13, WRK17, DKLs18] based on Oblivious Transfer (OT) extension [IKNP03, KOS15, PSS17, CSW20a, YWL+20] that only rely on cheap symmetric key operations and few public key operations. A recent line of works [YSWW21, BMRS21] presented interactive ZK protocols for circuits in the vector OLE (Oblivious Linear Evaluation) model [BCG+19, YWL+20]. Now that we know feasibility of RF for MPC via generic compilers, can we construct

RFs for efficient MPC protocols like those based on OT extension? All known techniques to construct RFs rely on some form of malleability/homomorphism of the underlying protocol so that the RF can randomize the messages. It is unclear how such randomization would work when the protocol messages are unstructured. Modifying the protocol to be homomorphic so as to be RF friendly defeats the purpose of protocols like OT extension where the goal is to minimize the number of public key operations. This motivates us to ask the following question:

Can we construct an MPC protocol in the reverse firewall setting where the number of public key operations is independent of the size of the circuit being computed?

We answer the above question in the affirmative by constructing such protocols for specific functions like OT extension and Zero-Knowledge (ZK). Constructing reverse firewalls for such protocols requires new techniques since the transcript resulting from symmetric key operations are unstructured and do not render themselves well to randomization.

1.1 Our Contributions

We initiate the study of efficient MPC protocols in the RF setting. Towards this end, we make the following contributions.

- We construct a variant of the KOS OT extension protocol [KOS15] together with an RF in the random oracle $\mathcal{F}_{\mathsf{RO}}$ model. Our protocol constructs $m = \mathsf{poly}(\kappa)$ correlated OT (cOT)[4] using only $\mathcal{O}(\kappa)$ public key operations. All prior constructions of maliciously secure OT [CDN20, CGPS21] require $\mathsf{poly}(\kappa)$ public key operations *per OT* due to their reliance on the GMW compiler and expensive ZK proofs. See Sec. 2.1, and 2.2 for an overview of our cOT functionality $\mathcal{F}_{\mathsf{cOT}}$ (in Fig. 1) and cOT extension protocol respectively.
- We construct a new base (random) OT protocol, which we use for our OT extension. In constructing the base OT protocol and RF (an overview of these ideas in Sec 2.3), we employ new ideas for malleable interactive proofs.
- We define a notion of *full malleability* for Sigma protocols that unlike prior notions allow randomizing the instance as well. We construct RFs for Sigma protocols and for OR composition that sanitize both the instance and the transcript. We show that ZK protocol resulting from the standard compilation of a Sigma protocol is fully malleable and construct an RF for it. These results could be of independent interest. We provide an overview of these ideas in 2.4.

Each base OT protocol require 35 exponentiations. For $\ell \leq \kappa$ base OTs in the OT extension, the cost of computing 35ℓ exponentiations gets amortized by generating $\mathsf{poly}(\kappa)$ extended cOTs. As a result each extended cOT communicates κ bits and computes roughly 4 symmetric key operations. Our correlated OT extension protocol in the firewall setting is captured in Thm. 1.

Theorem 1. *(Informal) Assume there exists an additively homomorphic commitment scheme Com, a collision resistant hash function H, a pseudorandom generator PRG, and that the Discrete Log assumption holds. We obtain a correlated OT extension protocol π_{cOT} with reverse firewalls that implements $\mathcal{F}_{\mathsf{cOT}}$ in $\mathcal{F}_{\mathsf{RO}}$-model when the honest parties' machines can be tampered and the adversary can maliciously corrupt either the sender or the receiver.*

We then show application of our cOT extension protocol in constructing efficient Zero-knowledge protocols. We build upon the recent interactive ZK protocol of Quicksilver [YSWW21] to obtain the first efficient ZK protocol for all of NP in the RF setting. We capture our contribution by the following theorem.

[4] Our cOT protocol allows the receiver to learn c bits of sender's secret with probability 2^{-c}. We capture this leakage in the ideal functionality $\mathcal{F}_{\mathsf{cOT}}$, and show that this weakened functionality suffices for constructing OT-based RF friendly ZK protocol.

Theorem 2. *(Informal) Assuming H is a collision resistant hash function and Com is an additively homomorphic commitment scheme, π_{QS} implements the Zero-knowledge $\mathcal{F}_{ZK}$ functionality in the $\mathcal{F}_{\mathsf{cOT}}$ model for NP in the presence of reverse firewalls where the honest parties' machines can be tampered and the adversary can maliciously corrupt either the prover or the verifier. Our construction requires $(n + t)$ invocations to $\mathcal{F}_{\mathsf{cOT}}$, where n is the number of input wires and t is the number of multiplication gates in the NP verification circuit for the statement.*

In π_{QS}, proving each multiplication gate requires one cOT, as in the original Quicksilver protocol. Instantiating $\mathcal{F}_{\mathsf{cOT}}$ with our π_{cOT} results in a proof size of $(n + t)\kappa$ bits. In comparison, the original Quicksilver implements $\mathcal{F}_{\mathsf{cOT}}$ using Silent-OT extension protocol [YWL+20] yielding a proof size of $(n + t)$ bits. We provide an overview of our Quicksilver variant and its RF in Sec 2.5.

Key Idea. The central idea of our work is to generate correlated data (cOTs in our case) between two parties to compute a circuit. We show how these correlated data can be generated from symmetric key operations and in an RF-friendly way. Previously, all RF-friendly techniques were for protocols relying on public-key primitives and the RF exploits the natural "structure". Our work shows that there is no inherent barrier for constructing RFs for protocols that rely on symmetric-key primitives. Concretely, we only need cheap symmetric key operations, and the number of public key operations (e.g. the base OTs) are independent of the size of the circuit to be computed. Looking ahead, this correlation allows the parties to verify a protocol transcript (e.g. the RF-compatible Quicksilver) efficiently. This verification can be performed using an inexpensive (solely based on symmetric key operations) consistency check. In contrast, if we were to use ZK proofs (GMW paradigm) for verification, RF-compatibility requires ZK to be controlled-malleable which are algebraic and inherently require public key operations. We believe our ideas to deal with unstructured data opens up a new paradigm for constructing more efficient RF-compatible protocols, especially as a stepping stone towards MPC protocols based on silent OT extension.

1.2 Future Work

Our current OT extension protocol can be used to construct MPC protocols in the RF setting using generic compilers [IPS08, NNOB12, DPSZ12] that rely on OTs. A natural extension of our work is to construct Silent OT extension family of protocols [BCG+19, YWL+20] in the RF setting. Current techniques in Silent OT extension paradigm requires the receiver to compute LPN samples and use them in the underlying bootstrapping protocol. In the RF setting, this is a non-trivial task since the LPN samples might be leaky due to bad randomness. It is not obvious how to sanitize them without relying on expensive public key operations or generic zero-knowledge. Our work shows that our correlated OTs suffice for designated-verifier ZK protocols. We believe that similar ideas could be instrumental in other designated-verifier settings, like silent-OT and authenticated garbling [YWZ20].

1.3 Related Work

Reverse firewalls. The work of [MS15] constructs RFs for a variant of the Naor-Pinkas OT protocol [NP01]. Their construction only provides passive security, whereas we are in the malicious setting. The work of [CMY+16] constructs an OT protocol from graded rings, incurring $\mathsf{poly}(\kappa)$ public key operations for each OT. While these works show feasibility, we focus on constructing OT extension protocols in the RF setting with malicious security, while retaining the advantage of OT extension – create $\mathsf{poly}(\kappa)$ OTs with *symmetric key operations* starting from κ base OTs. The other approaches via generic MPC compilers [CDN20, CGPS21] incur $\mathsf{poly}(\kappa)$ public key operations for each OT instance. In their original paper, Mironov and Stephens-Davidowitz [MS15] show how to construct reverse firewalls for oblivious transfer (OT) and two-party computation with semi-honest security. Follow-up research showed how to construct reverse firewalls for a plethora of cryptographic primitives and protocols including: secure message transmission and key agreement [DMS16, CMY+16], signature schemes [AMV15], interactive proof systems [GMV20], and maliciously secure MPC for both the case of static [CDN20] and adaptive [CGPS21] corruptions. The recent work

of [CMNV22] also introduced the notion of Universally Composable Subversion-Resilient security. Extending our results in their model is an interesting direction for future work.

As already mentioned in the introduction, all the above constructions use the ability of the RF to maul (in a controlled way) the transcript of the protocols to prevent exfiltration, which in turn required the underlying building blocks to be (controlled) homomorphic. Hence, the number of public key operations depends on the size of the circuit (representing the function) to be computed securely. This is in sharp contrast to our OT and (interactive) ZK protocols where the resulting protocols after RF sanitization performs a number of public key operations that are independent of the size of the circuit being computed.

Watchdogs. The line of research on *cliptography* [RTYZ16, CHY20, BCJ21] shows how to clip the power of subversion attacks without the need for reverse firewalls, and assuming only that a watchdog algorithm can perform a black-box test to decide whether a (possibly subverted) implementation is compliant to its specification.

A watchdog can be *offline* (meaning that testing only happens before a scheme is deployed), *online* (meaning that testing is executed in parallel to the deployment of the scheme), or *omniscient* (meaning that testing additionally depends on the implementation's secret state). Unfortunately, offline watchdogs alone are not powerful enough to detect pretty natural classes of subversion attacks such as input-triggered attacks [DFP15, AMV15]. In contrast, reverse firewalls allow to annihilate input-triggered attacks via offline testing *and* sanitation. Online watchdogs and omniscient watchdogs, instead, are both incomparable to reverse firewalls as the former needs to test a running implementation, and the latter needs access to secret inputs, but none of them sanitizes the protocol transcript.

Self-guarding. Fischlin and Mazaheri [FM18] proposed another defense mechanism called *self-guarding*, which requires users to have a trusted initialization phase to generate genuine outputs of a given cryptographic primitive. These outputs can later be used in order to sanitize the outputs produced by a possibly tampered implementation. The main advantage of this model is that it does not require an active party (such as the reverse firewall or the watchdog). The main disadvantage is that security depends on the number of samples collected during the initialization phase.

Backdooring. 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], non-interactive zero knowledge [BFS16], and public-key encryption [ABK18].

1.4 Backdooring of primitives vs Tampering of Implementations

Since our focus is on efficient MPC protocols and RFs, the RF-friendly protocols we construct are based on symmetric-key primitives like hash functions and pseudorandom generators (PRGs). While backdooring of such primitives is also of concern in the subversion setting, we argue that it is an issue orthogonal to the issue of *tampering of implementations* that we consider in this work. Backdooring of a primitive [BPR14, DGG+15] is subversion that poses a different kind of threat compared to tampering. In subversion, a powerful adversary (saboteur) chooses an instance of a primitive for which it knows a trapdoor. For example, the Dual EC PRG standardized by NIST is a family of PRGs parameterized by two group elements, and an entity that samples these parameters can remember the randomness used to sample which serves as a backdoor. This backdoor suffices to completely predict all outputs after seeing one output from the PRG. In general, subversion is concerned with powerful adversaries who *backdoor* a primitive. In contrast, the setting we consider is where a powerful adversary corrupts the computers of honest parties. In the protocols we construct, we design RFs to immunize the messages exchanged by the parties as part of the protocol, but assume that any primitives that the protocols might use (locally by the parties) are backdoorless. We also note that both prior works that construct RFs for MPC protocols [CDN20, CGPS21] are generic compilers and therefore also implicitly assume that all the primitives used by the underlying MPC protocol are backdoorless. We prove security of

our protocols in the Random Oracle Model (ROM), and it is known how to immunize backdoored primitives like PRGs in the ROM [DGG+15]. Once the RO is instantiated with a hash function like SHA-256, the assumption presumes that SHA-256 is itself not backdoored. The works of [FJM18, DFMT20] show how to immunize backdoored ROs and backdoored hash functions. Combining these immunization techniques with RFs to construct end-to-end solutions that address subversion is an interesting direction for future work. Countering both tampering of implementation and subversion of primitives simultaneously is important but not in the scope of this work.

Remark. Since our focus is on efficient MPC protocols and RFs, the RF-friendly protocols we construct are based on symmetric-key primitives like hash functions and Pseudorandom generators (PRGs). While backdooring of such primitives is also of concern in the subversion setting, we argue that it is an issue orthogonal to the issue of *tampering of implementations* that we consider in this work. We also note that both prior works that construct RFs for MPC protocols [CDN20, CGPS21] are generic compilers and therefore also implicitly assume that all the primitives used by the underlying MPC protocol are backdoorless. We provide a more detailed discussion comparing tampering of implementations and backdooring of primitives in Section 1.4. We prove security of our protocols in the Random Oracle (RO) model, and it is known how to immunize backdoored primitives like PRGs in the RO model [DGG+15]. Once the RO is instantiated with a hash function like SHA-256, the assumption presumes that SHA-256 is itself not backdoored. The works of [FJM18, DFMT20] show how to immunize backdoored ROs and backdoored hash functions. Combining these immunization techniques with RFs to construct end-to-end solutions that address subversion is an interesting direction for future work. Countering both tampering of implementation and subversion of primitives simultaneously is important but not in the scope of the current work.

2 Technical Overview

In this section, we discuss state-of-the-art protocols, some hurdles in adapting them to the RF setting and outline our techniques to overcome them. Our protocols are shown secure in the RF setting by relying on the recent result of [CGPS21], which showed that 1) if an MPC protocol satisfies simulation-based security, and 2) the firewall is functionality maintaining and provides exfiltration resistance, then the firewall preserves security of the protocol in the presence of functionality maintaining tampering. Thm. 3 in Sec. 3.2 formally summarizes the result. For every protocol we prove that it satisfies simulation-based security and their respective firewall provides exfiltration resistance. Combining Thm. 3 with simulation security and exfiltration resistance provides us the desired security guarantee.

2.1 Correlated OT with Leakage functionality

We initiate our overview discussion with the correlated OT functionality $\mathcal{F}_{\mathsf{cOT}}$ in Fig. 1 (taken from [KOS15]). It allows some leakage to a corrupt receiver. The receiver has a choice bit vector $\mathbf{b} \in \{0,1\}^{\ell}$. The functionality samples $\mathbf{s} \leftarrow_R \{0,1\}^{\kappa}$, $\mathbf{M} \leftarrow_R \{0,1\}^{\ell \times \kappa}$ and sets $\mathbf{Q}_j = \mathbf{M}_j \oplus (\mathbf{s} \odot b_j)$ for $j \in [\ell]$. The functionality sets $\mathbf{Q} = \{\mathbf{Q}_j\}_{j \in [j \in [\ell]]}$ and returns $\mathbf{M}$ to the receiver and the $(\mathbf{s}, \mathbf{Q})$ to the sender. The functionality allows the receiver to guess c bits of $\mathbf{s}$ and the receiver gets caught with $1 - 2^{-c}$ probability. We show that this weaker functionality suffices for the ZK protocol of Quicksilver [YSWW21].

2.2 Correlated Oblivious Transfer Extension in the RF setting

We use the KOS [KOS15] OT extension to implement the $\mathcal{F}_{\mathsf{cOT}}$ functionality. We recall the KOS protocol as follows:

Recalling KOS OT extension: In the KOS OT extension, the sender $\mathsf{S}_{\mathsf{Ext}}$ and receiver $\mathsf{R}_{\mathsf{Ext}}$ generate m ($= \mathsf{poly}(\kappa)$) OTs using κ invocations to the random OT functionality, i.e. $\mathcal{F}_{\mathsf{rOT}}$ [5](Fig. 9), (implemented by

[5] Each invocation of $\mathcal{F}_{\mathsf{rOT}}$ returns (a_0, a_1) to the sender and (b, a_b) to the receiver where $a_0, a_1 \leftarrow_R \{0,1\}^{\kappa}$ and $b \leftarrow_R \{0,1\}$ are randomly sampled by the functionality.

Functionality $\mathcal{F}_{\mathsf{cOT}}$

Upon receiving $(\textsc{Initiate}, \mathsf{sid}, \ell)$ from sender S and receiving $(\textsc{Initiate}, \mathsf{sid}, \mathbf{b}, \ell)$ from receiver R where $\mathbf{b} = (b_1, \ldots, b_\ell)$ and $b_j \in \{0,1\}$ for $j \in [\ell]$, the functionality $\mathcal{F}_{\mathsf{cOT}}$ interacts as follows:

- If S is corrupted receive $\mathbf{s} \in \{0,1\}^\kappa$ and $\mathbf{Q} \in \{0,1\}^{\ell \times \kappa}$ from the sender. Set $\mathbf{M}_j = \mathbf{Q}_j \oplus (\mathbf{s} \odot b_j)$ for $j \in [\ell]$.
- If R is corrupted then receive $\mathbf{M} \in \{0,1\}^{\ell \times \kappa}$ from the receiver, sample $\mathbf{s} \leftarrow_R \{0,1\}^\kappa$ and set $\mathbf{Q}_j = \mathbf{M}_j \oplus (\mathbf{s} \odot b_j)$ for $j \in [\ell]$.
- When a corrupt R guesses c bits of $\mathbf{s}$ by invoking $(\textsc{Guess}, \mathsf{sid}, \{\mathsf{ind}_i\}_{i\in[c]}, \{s'_i\}_{i\in[c]})$: $\mathcal{F}_{\mathsf{rOT}}$ aborts if $s'_i \neq s_{\mathsf{ind}_i}$ for any $i \in [c]$; otherwise all the guesses are correct and $\mathcal{F}_{\mathsf{rOT}}$ sends $(\textsc{Undetected}, \mathsf{sid})$ to $\mathcal{A}$.
- If both parties are honest, then sample $\mathbf{s} \leftarrow_R \{0,1\}^\kappa$, $\mathbf{M} \leftarrow_R \{0,1\}^{\ell \times \kappa}$ and set $\mathbf{Q}_j = \mathbf{M}_j \oplus (\mathbf{s} \odot b_j)$ for $j \in [\ell]$.

Denote $\mathbf{Q} = \{\mathbf{Q}_j\}_{j\in[\ell]}$ and $\mathbf{M} = \{\mathbf{M}_j\}_{j\in[\ell]}$. Send $(\mathsf{sent}, \mathsf{sid}, \mathbf{M})$ to R and $(\mathsf{sent}, \mathsf{sid}, (\mathbf{s}, \mathbf{Q}))$ to S and store $(\mathsf{sen}, \mathsf{sid}, \ell, (\mathbf{b}, \mathbf{M}, \mathbf{Q}))$ in memory. Ignore future messages with the same sid.

Fig. 1: Ideal functionality $\mathcal{F}_{\mathsf{cOT}}$ for Correlated Oblivious Transfer with leakage

base OTs) and symmetric key operations. In the base OTs, the sender $\mathsf{S}_{\mathsf{Ext}}$ plays the role of a receiver, and the receiver $\mathsf{R}_{\mathsf{Ext}}$ plays the role of a sender. The ith invocation of $\mathcal{F}_{\mathsf{rOT}}$ functionality returns random strings $(k_{i,0}, k_{i,1}) \leftarrow_R \{0,1\}^\kappa$ to the sender and (s_i, k_{i,s_i}) to the receiver where $s_i \leftarrow_R \{0,1\}$. The input of $\mathsf{R}_{\mathsf{Ext}}$ is bit string $\mathbf{r} \in \{0,1\}^m$ for m correlated extended-OTs. The receiver also samples κ random bits $\tau \leftarrow_R \{0,1\}^\kappa$ and sets $\mathbf{r}' = (\mathbf{r}||\tau) \in \{0,1\}^{m+\kappa}$. This is done to prevent leakage of input choice bits during the consistency checks. The receiver computes the choice bit matrix $\mathbf{R} \in \{0,1\}^{(m+\kappa)\times\kappa}$ where the jth row of $\mathbf{R}$ denoted as $\mathbf{R}_j$ is computed as follows:

$$\mathbf{R}_j = (r'_j, \ldots, r'_j) \text{ for } j \in [m+\kappa].$$

$\mathsf{R}_{\mathsf{Ext}}$ computes a matrix $\mathbf{M} \in \{0,1\}^{(m+\kappa)\times\kappa}$ such that the ith column of $\mathbf{M}$ denoted as $\mathbf{M}^i$ is computed as follows:

$$\mathbf{M}^i = \mathsf{PRG}(k_{i,0}) \text{ for } i \in [\kappa],$$

where $\mathsf{PRG} : \{0,1\}^\kappa \to \{0,1\}^{m+\kappa}$. $\mathsf{R}_{\mathsf{Ext}}$ sends a mapping $\mathbf{D}$ from his choice bits $\mathbf{r}' \in \{0,1\}^m$ to the $(\mathbf{k}_{i,0}, \mathbf{k}_{i,1})$ values. The ith column of $\mathbf{D}$ is denoted as $\mathbf{D}^i$ and is computed as follows:

$$\mathbf{D}^i = \mathsf{PRG}(k_{i,0}) \oplus \mathsf{PRG}(k_{i,1}) \oplus \mathbf{R}^i \text{ for } i \in [\kappa].$$

Upon obtaining this mapping $\mathbf{D}$ and the base-OT output, the sender computes his mapping as $\mathbf{Q}$ where the ith column of $\mathbf{Q}$ is denoted as follows:

$$\mathbf{Q}^i = \left(s_i \odot \mathbf{D}^i\right) \oplus \mathsf{PRG}(k_{i,s_i}) \text{ for } i \in [\kappa],$$

The jth row of $\mathbf{Q}$ denoted as $\mathbf{Q}_j$ satisfies the following relation:

$$\mathbf{Q}_j = \mathbf{M}_j \oplus (\mathbf{s} \odot \mathbf{R}_j) = \mathbf{M}_j \oplus (\mathbf{s} \odot r_j) \text{ for } j \in [m].$$

In addition to the above, the sender performs consistency checks [Dia22]. A corrupt receiver can leak bits of $\mathbf{s}$ if the rows of $\mathbf{R}$ are not monochrome, i.e. $\exists j \in [m]$ s.t. $\mathbf{R}_j$ is neither 0^κ nor 1^κ. Such an attack can be launched by the corrupt receiver if $\mathbf{D}$ is malformed. To detect such malicious behaviour, the sender performs a consistency check on matrix $\mathbf{D}$. In the original KOS paper, the protocol consists of an interactive check phase. The receiver and sender perform a coin-tossing protocol to generate $m+\kappa$ fields elements $\chi \leftarrow_R \mathbb{F}^{m+\kappa}$ using a random oracle $\mathcal{F}_{\mathsf{RO}}$, where $\mathbb{F} = \mathcal{O}(2^\mu)$ and μ is the statistical security parameter. The receiver computes $\mathbf{u}$ and $\mathbf{v}$ as part of the consistency check on $\mathbf{D}$:

$$\mathbf{u} = \bigoplus_{j\in(m+\kappa)} (\chi_j \cdot \mathbf{M}_j), \mathbf{v} = \bigoplus_{j\in(m+\kappa)} (\chi_j \cdot \mathbf{R}_j)$$

The receiver sends $(\mathbf{u}, \mathbf{v})$ to the sender as the response of the consistency checks. The sender computes $\mathbf{w}$ as follows:

$$\mathbf{w} = \bigoplus_{j \in (m+\kappa)} (\chi_j \cdot \mathbf{Q}_j).$$

The sender aborts if $\mathbf{w} \neq \mathbf{u} \oplus \mathbf{s} \cdot \mathbf{v}$. The consistency checks ensure that the receiver learns only c bits of $\mathbf{s}$ with probability 2^{-c} probability. We follow the same approach. Once the consistency checks pass, the receiver sets $\{r_j, \mathbf{M}_j\}$ as the output of the jth cOT for $j \in [m]$. The sender sets $(\mathbf{s}, \mathbf{Q}_j)$ as the output of the jth cOT.

Obstacles in RF setting and key insights. The above protocol fails to provide exfiltration resistance in the RF setting. We highlight the problems and outline solution ideas.

- **Implementing $\mathcal{F}_{\mathsf{cOT}}$:** There is no protocol π_{rOT} in MPC literature that implements $\mathcal{F}_{\mathsf{rOT}}$ functionality while providing ER for tampered honest parties. In order to provide ER, the firewall needs to rerandomize the OT protocol transcript such that the receiver's choice bit gets randomized and the sender's messages are rerandomized. The state-of-the-art OT protocols of [PVW08, CSW20b] are in the setup string model where the setup string can be tampered. Moreover, a firewall cannot rerandomize the first message of the receiver to rerandomize the receiver's choice bit since the tampered receiver would then be unable to decrypt the sanitized OT transcript. Meanwhile, the protocols of [BPRS17, MR19, CSW20a] are in the random oracle model where the messages in the OT transcript consists of random oracle outputs. It is unclear how a firewall could rerandomize such transcripts since it would require computing the preimage of the random oracle output. To address this issue, we build a new base OT protocol π_{rOT} which implements $\mathcal{F}_{\mathsf{rOT}}$ functionality and provides exfiltration resistance for tampered parties. Overview of the base OT protocol is discussed in Sec. 2.3.
- **Rerandomizing D matrix:** A malicious receiver could send a "signal" such that a tampered sender behaves differently thereby leaking one bit of the honest (tampered) sender's input. For instance, a malicious receiver can choose its choice bits $\mathbf{r}$ in a way such that $\mathbf{D}$ lies in a particular distribution (e.g. the first column of $\mathbf{D}$ is all 0s). A tampered sender aborts upon receiving this malformed $\mathbf{D}$ matrix while an honest sender does not. This leaks one bit of the sender's input violating exfiltration resistance. We address this issue by using a technique such that the $\mathbf{r}$ vector is randomly chosen as part of the protocol. The receiver and the sender perform an augmented coin-tossing protocol where the receiver obtains random coins coin and the sender obtains a commitment to the coin as c_{coin}. The receiver generates the first column of $\mathbf{D}$, denoted as $\mathbf{D}^1$, by invoking the random oracle $\mathcal{F}_{\mathsf{RO}}$ on coin. The receiver is required to compute the choice bit vector(and the padding bits) $\mathbf{r}' = \mathbf{r}||\tau$ from $\mathbf{D}^1$ and the outputs of the base OTs as follows:

  $$\mathbf{r}' = \mathbf{D}^1 \oplus \mathsf{PRG}(k_{1,0}) \oplus \mathsf{PRG}(k_{1,1})$$

  This rerandomizes the $\mathbf{r}'$ vector, and as a result the $\mathbf{D}$ matrix cannot be used to exfiltrate by choosing a tampered choice bit vector $\mathbf{r}$ (or τ). The receiver is required to decommit to c_{coin} when it sends $\mathbf{D}$ to the sender. The sender verifies the opening and also verifies that the first column of $\mathbf{D}$ is generated by invoking the random oracle $\mathcal{F}_{\mathsf{RO}}$ on coin.
- **Consistency Checks:** A malicious receiver can still send a badly constructed $\mathbf{D}$ (rows of the computed $\mathbf{R}$ are not monochrome) which might trigger a tampered sender. Upon obtaining $\mathbf{D}$ the tampered sender can abort thus leaking one bit of its input. In contrast, an honest sender does not abort until the end of the consistency checks. This behaviour could exfiltrate secrets of a tampered sender to a malicious receiver. We observe that if a malicious receiver sends a malformed $\mathbf{D}$ and the consistency check is performed correctly then a tampered sender aborts, similar to an honest sender, since the tampering is functionality maintaining. However, the sender should obtain $\mathbf{D}$ and the receiver's response for the consistency check in the same round. In such a case, an honest sender also aborts if $\mathbf{D}$ is malformed as this is detected in the consistency check. A tampered sender also aborts and now this prevents exfiltration even if $\mathbf{D}$ contains hidden triggers since the abort is due to the checks failing. The behaviour of the tampered sender is statistically indistinguishable from an honest sender: they only differ when the checks fail to

detect inconsistency which occurs with probability $\frac{1}{|\mathbb{F}|}$. This observation leads us to a modified protocol such that it provides ER for a tampered sender.

After computing the base OTs, the corrupt receiver commits to the hash of $\mathbf{D}$ using an additively homomorphic commitment scheme as $c_{\mathbf{D}}$. Additive homomorphism allows rerandomization of the commitment by the firewall. The parties then generate the coins seed for the consistency check using an RF-compatible augmented coin-tossing protocol. The randomness for the consistency checks are derived from $\mathcal{F}_{\mathsf{RO}}(\mathsf{seed})$, where $\mathcal{F}_{\mathsf{RO}}$ is the random oracle. Finally, the receiver sends $\mathbf{D}$, the decommitment of $c_{\mathbf{D}}$ to $H(\mathbf{D})$ and the response to the consistency checks. The sender verifies the decommitment and the response to the consistency check.

The hash function and $c_{\mathbf{D}}$ forces a corrupt receiver to succinctly commit to $\mathbf{D}$ and allows it to decommit to $\mathbf{D}$ along with the response to the consistency check. The consistency check forces a tampered sender to abort if $\mathbf{D}$ is malformed in a way oblivious to any hidden triggers. This provides exfiltration resistance for the sender. The commitment $c_{\mathbf{D}}$ is rerandomized by the firewall. seed is rerandomized by the firewall to $\widehat{\mathsf{seed}} = \mathsf{seed} + \widetilde{\mathsf{seed}}$. To incorporate $\widetilde{\mathsf{seed}}$ into c_{seed} the firewall computes $\widehat{c_{\mathsf{seed}}} = c_{\mathsf{seed}} \cdot \mathsf{Com}(\mathsf{seed}; \widetilde{\delta_{\mathsf{seed}}})$ and sends $\widehat{c_{\mathsf{seed}}}$ to receiver on behalf of sender. The firewall also sends $\widehat{c_{\mathsf{R}}} = \mathsf{seed}_{\mathsf{R}} + \widetilde{\mathsf{seed}}$ to the sender on behalf of the receiver. When sender opens c_{seed} to $(\mathsf{seed}_{\mathsf{S}}; \delta_{\mathsf{seed}})$ the firewall sends $(\mathsf{seed}_{\mathsf{S}} + \widetilde{\mathsf{seed}}, \delta_{\mathsf{seed}} + \widetilde{\delta_{\mathsf{seed}}})$ to the receiver. This ensures that both parties obtain the coins as $\widehat{\mathsf{seed}}$. We also assume that the commitments are additively homomorphic so that they can be rerandomized by the firewall. The only way to tamper $\mathbf{D}$ matrix and not get caught is when the receiver guesses κ bits of $\mathbf{s}$ to pass the consistency checks. However, the checks ensure that such an event occurs with $2^{-\kappa}$ probability.

The protocol with the three changes gives us a correlated OT (with leakage) extension protocol π_{cOT}. The protocol is presented in Fig. 7 and the firewall in Fig. 8. Our correlated OT with leakage is weaker than correlated OT of [YSWW21] since it allows a corrupt receiver to compute c bits of sender's secret key $\mathbf{s}$ with probability 2^{-c}. However, as we show in Sec. 2.5, this suffices for Quicksilver [YSWW21]. Next, we build our base OT protocol π_{rOT} which implements $\mathcal{F}_{\mathsf{rOT}}$.

2.3 Base Oblivious Transfer Protocols in the RF setting

As discussed above, the state-of-the-art OT protocols [PVW08, MR19, CSW20b, CSW20a] fail to give π_{rOT} in the presence of functionality maintaining tampering. The OT protocol of [MS15] provides only passive security in the RF setting and no guarantees against active corruption of the receiver. We construct π_{rOT} by building upon the classical OT protocol of [BM90] in the plain model. For the sake of completeness, we first recall the protocol.

Protocol of [BM90]. The sender samples a field element q and computes group element $Q = g^q$ and sends Q to the receiver. The receiver has a choice bit b and it samples two public keys $(\mathsf{pk}_0, \mathsf{pk}_1)$ such that $\mathsf{pk}_b = g^{\mathsf{sk}}$ for secret key sk and $Q = \mathsf{pk}_0 \cdot \mathsf{pk}_1$. The receiver sends pk_0 to the sender. The sender samples $r_0, r_1 \leftarrow_R \mathbb{Z}_q$, and computes $R_0 = g^{r_0}$ and $R_1 = g^{r_1}$. The sender sets the output as $k_0 = H(\mathsf{pk}_0^{r_0})$ and $k_1 = H(\mathsf{pk}_1^{r_1})$ where H is the Goldreich-Levin hash function or a random oracle. The sender sends (R_0, R_1) to the receiver. The receiver outputs $k_b = H(R_b^{\mathsf{sk}})$.

Modifications for Simulation-based security. The protocol of [BM90] only provides semantic security. The receiver's choice bit is perfectly hidden in the first message pk_0 and the sender's messages are (k_0, k_1) not extractable. We make the following changes in order to allow for simulation based security:

– **Sender Input Extraction:** To extract the sender's input, we modify the protocol so that the sender proves knowledge of q such that $Q = g^q$ through an interactive protocol zero knowledge proof of knowledge (ZKPOK) with the receiver as the verifier. The simulator extracts q from the ZKPOK and sets the secret keys as $\mathsf{sk}_0 \leftarrow_R \mathbb{Z}_q$ and $\mathsf{sk}_1 = q - \mathsf{sk}_0$. The knowledge of the two secret keys enables the simulator to extract the corrupt sender's outputs (k_0, k_1). The ZK property of the proof ensures that q is hidden from a corrupt receiver.

– **Receiver Input Extraction:** To extract the receiver's input, we modify the protocol so that the receiver proves knowledge of sk for the statement $((\mathsf{pk}_0, \mathsf{pk}_1, \mathbb{G}, \mathbb{Z}_q) : \exists \mathsf{sk} \in \mathbb{Z}_q, b \in \{0,1\}$ s.t. $(\mathsf{pk}_0 = g^{\mathsf{sk}} \vee \mathsf{pk}_1 = g^{\mathsf{sk}}))$ using a Witness indistinguishability proof of knowledge (WIPOK). The simulator extracts (sk, b) from the WI proof. Meanwhile, the simulator against a corrupt sender is able to simulate the proof by setting $\mathsf{pk}_0 = g^{\mathsf{sk}}$ and $b = 0$ by relying on the WI property. We also set $k_0 = \mathsf{pk}_0^{r_0}$ and $k_1 = \mathsf{pk}_1^{r_1}$ for efficiency purposes and remove the Goldreich-Levin hash function. The WI proof ensures that if the proof accepts then the receiver has full knowledge of (sk, b). Using the knowledge of sk, we reduce a corrupt receiver breaking semantic security of the OT scheme to an adversary breaking DDH.

Modifications in RF setting. The above protocol fails to provide exfiltration resistance. We highlight some problems and suggest solutions.

– **Rerandomizing OT parameter Q:** A malicious sender can malform Q and use it as a trigger for a tampered receiver. To address this issue, we generate Q using coin tossing where the receiver sends $T = \mathsf{Com}(Q_{\mathsf{R}})$ and the sender sends a share Q_{S}. The receiver later decommits to Q_{R} and both parties set $Q = Q_{\mathsf{R}} \cdot Q_{\mathsf{S}}$ as the parameter. A firewall can sanitize this: sample $\widetilde{q} \leftarrow_R \mathbb{Z}_q, \widetilde{t} \leftarrow_R \{0,1\}^*$ and sanitize the commitment as $\widehat{T} = T \cdot \mathsf{Com}(g^{\widetilde{q}}; \widetilde{t})$ and sanitize Q_{S} as $\widehat{Q_{\mathsf{S}}} = Q_{\mathsf{S}} \cdot g^{\widetilde{q}}$ such that the new parameter is $\widehat{Q} = Q \cdot g^{\widetilde{q}}$ where $\widetilde{q} \leftarrow_R \mathbb{Z}_q$. The firewall also invokes the firewall of the ZK protocol with instance rerandomizer $\widetilde{q}$ since the receiver produces a ZK proof for $(Q_{\mathsf{S}}, \mathbb{G}, \mathbb{Z}_q)$ and the firewall sanitizes it to a proof of $(Q_{\mathsf{S}} \cdot g^{\widetilde{q}}, \mathbb{G}, \mathbb{Z}_q)$. More discussion about the ZK firewall can be found in Sec. 2.4. This transformation provides ER to both parties corresponding to the OT parameters and the ZK proof.

– **Rerandomizing Receiver's choice bit and public keys:** The firewall needs to rerandomize the receiver's choice bit and the public keys to implement $\mathcal{F}_{\mathsf{rOT}}$ functionality and prevent exfiltration through the public keys. To enable this, we have the sender commit to a pad $p \leftarrow_R \mathbb{Z}_q$ using an additively homomorphic commitment as $c^{\mathsf{S}} = \mathsf{Com}(p; d^{\mathsf{S}})$. When the sender receives $(\mathsf{pk}_0, \mathsf{pk}_1)$ the sender decommits to p and the receiver sets the new public keys as $\mathsf{pk}'_0 = \mathsf{pk}_0 \cdot g^p$ and $\mathsf{pk}'_1 = \mathsf{pk}_1 \cdot g^{-p}$. These new public keys maintain the invariant that $\mathsf{pk}'_0 \cdot \mathsf{pk}'_1 = Q$. The firewall sanitizes the public keys by changing p to $\widehat{p} = p + \widetilde{p}$. The commitment is modified to $\widehat{c^{\mathsf{S}}} = c^{\mathsf{S}} \cdot \mathsf{Com}(\widetilde{p}; \widetilde{d^{\mathsf{S}}})$. Upon receiving the decommitment (p, d^{S}) the firewall modifies it to $(\widehat{p}, d^{\mathsf{S}} + \widetilde{d^{\mathsf{S}}})$. Upon receiving the public keys $(\mathsf{pk}_0, \mathsf{pk}_1)$ the firewall changes it to $(\mathsf{pk}_0 \cdot g^{\widetilde{p}}, \mathsf{pk}_1 \cdot g^{\widetilde{-p}})$. This allows both parties to get sanitized public keys $(\widehat{\mathsf{pk}_0}, \widehat{\mathsf{pk}_1}) = (\mathsf{pk}_0 \cdot g^{p+\widetilde{p}}, \mathsf{pk}_1 \cdot g^{-p-\widetilde{p}})$. It is ensured that $\widehat{\mathsf{pk}_0} \cdot \widehat{\mathsf{pk}_1} = \widehat{Q}$ thus preventing any exfiltration through the public keys. Next, we rerandomize the choice bit of the receiver where the sender sends a random bit ρ in the last message of the OT protocol. The receiver's new choice bit is set to $s = b \oplus \rho$ where b was initially chosen by the receiver by sampling $\mathsf{sk} \leftarrow_R \mathbb{Z}_q$ and setting $\mathsf{pk}_b = g^{\mathsf{sk}}$. The firewall sanitises ρ to $\widehat{\rho} = \rho \oplus \widetilde{\rho}$ and it permutes the order of pk_0 and pk_1 if $\widetilde{\rho} = 1$. The firewall also modifies the commitment c^{seed} accordingly so that the order of the sanitised public keys are consistent for both parties. Finally, these changes are also reflected in the WIPOK proof performed by the receiver as the prover. Recall that the receiver proves knowledge of witness for the statement $((\mathsf{pk}_0, \mathsf{pk}_1, \mathbb{G}, \mathbb{Z}_q) : \exists w \in \mathbb{Z}_q, b \in \{0,1\}$ s.t. $(\mathsf{pk}_0 = g^w \vee \mathsf{pk}_1 = g^w))$ using a WIPOK. The firewall sanitizes the proof such that it is consistent with the sanitized public keys and the order of the keys. In particular, if $\widetilde{\rho} = 0$ the new statement is $((\mathsf{pk}_0 \cdot g^{\widetilde{p}}, \mathsf{pk}_1 \cdot g^{-\widetilde{p}}, \mathbb{G}, \mathbb{Z}_q) : \exists w \in \mathbb{Z}_q, b \in \{0,1\}$ s.t. $(\mathsf{pk}_0 \cdot g^{\widetilde{p}} = g^w \vee \mathsf{pk}_1 \cdot g^{-\widetilde{p}} = g^w))$. If $\widetilde{\rho} = 1$ the new statement is $((\mathsf{pk}_0 \cdot g^{\widetilde{p}}, \mathsf{pk}_1 \cdot g^{-\widetilde{p}}, \mathbb{G}, \mathbb{Z}_q) : \exists w \in \mathbb{Z}_q, b \in \{0,1\}$ s.t. $(\mathsf{pk}_0 \cdot g^{\widetilde{p}} = g^w \vee \mathsf{pk}_1 \cdot g^{-\widetilde{p}} = g^w))$. This is performed by constructing malleable Interactive WIPOKs in the RF setting where the *instance* is also sanitized. The firewall for the OT protocol invokes the WI RF with input $((\widetilde{p}, -\widetilde{p}), \widetilde{\rho})$. Detailed discussion about WI is in Sec. 2.4.

– **Rerandomizing sender's messages:** Finally the sender's pads (R_0, R_1) for the OT protocol needs to be rerandomized to implement $\mathcal{F}_{\mathsf{rOT}}$ functionality. The receiver commits to $(v_0, v_1) \leftarrow_R \mathbb{Z}_q$ and sends the commitments alongwith the public keys. Upon receiving (R_0, R_1), the receiver opens to (v_0, v_1) and considers the sender's random pads as $(R_0 \cdot g^{v_0}, R_1 \cdot g^{v_1})$. The sender sets the new randomness as $(r_0 + v_0, r_1 + v_1)$. The firewall sanitizes the commitment and the interaction such that the random pads are $(R_0 \cdot g^{v_0} \cdot g^{\widetilde{v_0}}, R_1 \cdot g^{v_1} \cdot g^{\widetilde{v_1}})$ and the sender's randomness are $(r_0 + v_0 + \widetilde{v_0}, r_1 + v_1 + \widetilde{v_1})$. This ensures that the tampered sender's pads are indistinguishable from an honestly generated sender random pads.

We obtain our base OT protocol implementing $\mathcal{F}_{\mathsf{rOT}}$ in Fig. 10 and 11. by carefully putting together the above ideas. While this overview is for $\ell = 1$ for simplicity, the final protocol implements $\mathcal{F}_{\mathsf{rOT}}$ for general ℓ. The protocol and the firewall are presented in Sec. 5. Each OT instance communicates 13 group elements + 15 field elements + 1 bit, and performs 35 exponentiations. In comparison, previous maliciously secure OT protocols [CDN20, CGPS21] rely on the GMW compiler, and compute $\mathsf{poly}(\kappa)$ exponentiations and communicate $\mathsf{poly}(\kappa)$ bits.

2.4 Malleable Interactive Protocols in the RF setting

We consider a class of interactive protocols based on Sigma protocols. For the sake of concreteness, consider the classical Sigma protocol for proving knowledge of a discrete logarithm [Sch90]. The statement consists of the description of a cyclic group $\mathbb{G}$ of prime order q, a generator g and an instance $x = g^w$, for $w \in \mathbb{Z}_q$. The prover's first message is a random group element $a = g^\alpha$. For a verifier's challenge $c \in \mathbb{Z}_q$, the prover's response is $z = a + wc$. The transcript $\tau = (a, c, z)$ is accepting if $g^z = ax^c$. We need to rerandomize the transcript without breaking the completeness condition, and without knowing the witness. In addition, since we use these interactive protocols in constructing our OT protocol, the instance x could also potentially be subliminal and therefore, we need to randomize the instance as well, to generate a randomized transcript $(\hat{x}, \hat{\tau})$. In order to build RFs for the ZK protocol obtained by compiling a Sigma protocol, we need to sanitize additional messages. Here, we rely on the key and message homomorphism of the Pedersen commitment scheme to randomize the commitment key, the commitment, and the message inside the commitment. Finally, we construct an RF for the OR composition that not only randomizes each instance in the compound statement, but the *entire* statement (by permuting the clauses). This is necessary since we use the OR protocol as a building block in a larger protocol where the statement itself could be tampered and needs to be sanitized.

We emphasize that our RFs randomize not just the transcript (a, c, z), but also the instance x, as opposed to the RF constructions in [GMV20] where the sanitized transcript still verifies for the same instance x. In our setting, crucially, the instance could also potentially be subliminal and therefore, needs to be randomized to prevent exfiltration. Our notion of fully malleable Sigma protocol is stronger than the malleability considered in [GMV20].

2.5 Efficient Zero-Knowledge in the RF setting

The recent works of [YSWW21, BMRS21] present interactive ZK protocols for circuits in the vector OLE model [BCG+19, YWL+20]. We focus on the work of Quicksilver [YSWW21] for binary circuits. In this setting, the vector OLE over binary field is modeled by the $\mathcal{F}_{\mathsf{cOT}}$ functionality. In Quicksilver, the parties run an interactive preprocessing phase which depends only on the security parameter. The parties obtain correlated randomness through this phase. In the online phase the prover obtains the NP verification circuit C and the witness wire assignment $\mathbf{w}$. The verifier obtains the circuit C. The parties locally expand their correlated randomness. The prover obtains $\mathbf{M} \in \{0,1\}^{\ell \times \kappa}$ and a random $\mathbf{b} \in \{0,1\}^\ell$, the verifier obtains $\mathbf{K} \in \{0,1\}^{\ell \times \kappa}$ and a random $\Delta \in \{0,1\}^\kappa$ such that the following holds for $i \in [\ell]$, where $\mathbf{K} = \{K_i\}_{i \in [\ell]}$, $\mathbf{M} = \{M_i\}_{i \in [\ell]}$, $\mathbf{b} = \{b_i\}_{i \in [\ell]}$:

$$K_i = M_i \oplus b_i \odot \Delta$$

Assume that the number of input wires to the circuit is n, the number of multiplication gates is t and $\ell = n + t$. The prover commits to the $n + t$ wire assignments for the input wires and multiplication gates by sending the mapping $d_i = w_i \oplus b_i$ to the verifier. Addition gates are free due to additive homomorphism and can be verified locally. The verifier updates K_i as follows for $i \in [n + t]$:

$$K_i = K_i \oplus d_i \odot \Delta = (M_i \oplus b_i \odot \Delta) \oplus (w_i \oplus b_i) \odot \Delta = M_i \oplus w_i \odot \Delta.$$

The prover P proves that the committed values w_i corresponding to the multiplication gates are correct by executing a batched verification phase with the verifier V. For each multiplication gate (α, β, γ) with input

wires α and β and output wire γ, the prover P has (w_α, M_α), (w_β, M_β), (w_γ, M_γ) and the verifier V holds $K_\alpha, K_\beta, K_\gamma, \Delta$ such that the following four equations should hold:

$$w_\gamma = w_\alpha \cdot w_\beta \quad \text{and} \quad M_i = K_i \oplus w_i \odot \Delta \ \text{ for } i \in \{\alpha, \beta, \gamma\}.$$

This can be verified by the verifier by performing the following check where prover sends $A_{i,0}$ and $A_{i,1}$:

$$\overbrace{B_i = K_\alpha \cdot K_\beta \oplus K_\gamma \cdot \Delta}^{\text{known to } \mathsf{V}} \stackrel{?}{=} \overbrace{M_\alpha \cdot M_\beta + (w_\beta \cdot M_\alpha \oplus w_\alpha \cdot M_\beta \oplus M_\gamma)}^{\text{known to } \mathsf{P}} \cdot \overbrace{\Delta}^{\text{known to } \mathsf{V}}$$
$$= A_{i,0} \oplus A_{i,1} \cdot \Delta$$

A corrupt prover passes the check even if $w_\gamma \neq w_\alpha \cdot w_\beta$ if it correctly guesses Δ, which occurs with $2^{-\kappa}$ probability. This covers the case for one gate. To check t multiplication gates in a batch the verifier sends a challenge χ. The prover and verifier also generates a random linear relationship $B^* = A_0^* \oplus A_1^* \cdot \Delta$ to mask the prover's inputs. This is performed using additional κ cOTs. The prover computes (U, V) as described below. The prover sends (U, V) and the verifier locally computes W.

$$U = \bigoplus_{i \in [n+t]} A_{i,0} \oplus A_0^* \quad , \quad V = \bigoplus_{i \in [n+t]} A_{i,1} \oplus A_1^* \quad , \quad W = \bigoplus_{i \in [n+t]} B_i \oplus B^*$$

The verifier outputs accept if $(W == U \oplus V \cdot \Delta)$ and rejects the proof if the equation fails to satisfy. A corrupt prover successfully cheats in the batch verification with probability $2^{-\kappa}$ by guessing Δ. Meanwhile, the ZK simulator simulates the proof by passing the check, given the knowledge of $(\mathbf{K}, \Delta)$ from $\mathcal{F}_{\mathsf{cOT}}$. The simulator computes W, samples $V \leftarrow_R \{0,1\}^\kappa$ and sets $U = W \oplus V \cdot \Delta$.

Modifications in RF setting. In order to achieve ER in the firewall setting, we make the following changes to the above protocol.

- **Preprocessing Phase:** The above protocol provides ER for the preprocessing phase if we implement $\mathcal{F}_{\mathsf{cOT}}$ with π_{cOT} with parameter $\ell = n+t+\kappa$. However, the parties need to know the number of extended correlated OTs (i.e. $n + t + \kappa$) in π_{cOT} during the preprocessing phase and perform communication proportional to it.
- **Batch Verification:** The mappings $\mathbf{d}$ maybe malformed and can be used to leak $\mathbf{w}$. Similarly, the challenge χ maybe malformed and can be used by a malicious verifier to trigger a tampered prover. We address these issues by following an approach similar to the consistency check in π_{cOT}. The prover commits to hash of $\mathbf{d}$ as $c_{\mathbf{d}}$. Upon receiving the commitment, the parties participate in an interactive coin tossing protocol to generate the challenge χ. Upon receiving the challenge, the prover decommits $\mathbf{d}$ and computes the response to the batch verification (following the original Quicksilver protocol). The verifier checks the decommitment to $\mathbf{d}$ and performs the verifier algorithm of the original quicksilver protocol. The soundness argument of the check is preserved if the hash is collision resistant, $c_{\mathbf{d}}$ is instantiated using a binding commitment scheme and the coin-tossing returns a random χ in the presence of functionality maintaining tamperings. The coin-tossing subprotocol is same as the coin tossing protocol in π_{cOT}. The firewall construction is also the same and this ensures ER for the coin-tossing. We refer to the *Consistency Checks* Sec. 2.2 for the discussion on the coin-tossing. Given that Δ is random and the challenge is sanitized by the firewall, a corrupt prover gets caught if $\mathbf{d}$ vector is malformed such that the underlying $\mathbf{w} = \mathbf{b} \oplus \mathbf{d}$ is invalid, i.e. $C(\mathbf{w}) = 0$. The complete protocol π_{QS} cna be found in Sec. 7.

The original quicksilver paper achieves communication complexity of 1 bit per multiplication gate. We incur a cost of $\kappa(1 + o(1)) < 2\kappa$ bits per multiplication gate. The number of public key operations is $\mathcal{O}(\kappa)$. The prover and verifier can run our protocol to verify a batch of m different circuits $(C_1, C_2, \ldots, C_m)$ with parameters $(\ell_1, \ell_2, \ldots, \ell_m)$ where ℓ_i denotes the number of input wires and multiplication gates in C_i. In such a case the parties invoke $\mathcal{F}_{\mathsf{cOT}}$ with parameter $L = \Sigma_{i \in [m]} \ell_i$. The number of public key operations for the base OTs gets amortized over m runs of the ZK protocol.

3 Preliminaries

Notations: We denote by $a \leftarrow D$ a uniform sampling of an element a from a distribution D. The set of elements $\{1, \ldots, n\}$ is represented by $[n]$. We denote the computational security parameter by κ and statistical security parameter by μ respectively. Let $\mathbb{Z}_q$ denote the field of order q, where $q = \frac{p-1}{2}$ and p are primes. Let $\mathbb{G}$ be the multiplicative group corresponding to $\mathbb{Z}_p^*$ with generator g, where CDH assumption holds. We denote a field of size $\mathcal{O}(2^\mu)$ as $\mathbb{F}$. For a bit $b \in \{0,1\}$, we denote $1-b$ by $\bar{b}$. We denote a matrix by $\mathbf{M}$ and let $\mathbf{M}^i$ refer to the ith column and $\mathbf{M}_j$ to the jth row of $\mathbf{M}$ respectively. Given a field element $x \in \mathbb{F}$ and a bit vector $\mathbf{a} = (a_1, a_2, \ldots, a_\kappa)$ we write component-wise multiplication as $x \cdot \mathbf{a} = (a_1 \cdot x, a_2 \cdot x, \ldots, a_\kappa \cdot x)$. Given two vectors $\mathbf{a}, \mathbf{b} \in \{0,1\}^n$, we denote component-wise multiplication by $\mathbf{a} \odot \mathbf{b} = (a_1 \cdot b_1, \ldots, a_n \cdot b_n)$.

3.1 Commitment Schemes

We define an additively homomorphic non-interactive commitment scheme Com as a tuple of two algorithms $(\mathsf{Gen}, \mathsf{Com})$ such that Com satisfies the following properties:

Definition 1. (Computationally Binding) *Com is computationally binding scheme if the following holds true for all PPT adversary $\mathcal{A}$:*

$$\Pr\left[(m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(pp) | pp \leftarrow \mathsf{Gen}(1^\kappa),\right.$$

$$\left.\mathsf{Com}(pp, m_0; r_0) = \mathsf{Com}(pp, m_1; r_1)\right] \leq \mathsf{neg}(\kappa)$$

Definition 2. (Computationally Hiding) *Com is a computationally hiding scheme if the following holds true for all PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:*

$$\Pr\left[b = b' | pp \leftarrow \mathsf{Gen}(1^\kappa), (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(pp), b \leftarrow \{0,1\},\right.$$

$$\left.c \leftarrow \mathsf{Com}(pp, m_b; r), b' \leftarrow \mathcal{A}_2(c; \mathsf{st})\right] \leq \frac{1}{2} + \mathsf{neg}(\kappa)$$

Definition 3. (Additively Homomorphic Commitment) *A commitment scheme $\mathsf{Com} = (\mathsf{Gen}, \mathsf{Com})$ is* additively homomorphic *over message space $\mathcal{M}$ and randomness space $\mathcal{R}$, which are written additively, such that for all $m, m' \in \mathcal{M}$, $r, r' \in \mathcal{R}$ we have:*

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

In our protocols we assume that the public parameters are implicitly provided to the parties to avoid notation overloading. We denote an additively homomorphic commitment scheme over $\mathbb{G}$ and $\mathbb{Z}_q$ as $\mathsf{Com}_\mathbb{G}$ and Com_q respectively. We show to instantiate such a commitment scheme from discrete log and DDH assumptions below.

Pedersen Commitment Scheme. Given generators $g, h \in \mathbb{G}$, the committer commits to a field element $m \in \mathbb{Z}_q$ 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, computationally binding due to the Discrete Log assumption and satisfies additive homomorphism in a straightforward manner.

Elgamal Commitment Scheme. We present a version [CGPS21] of Elgamal commitment scheme without setup as follows. Given a generator $g \in \mathbb{G}$, the committer commits to a group element $m \in \mathbb{G}$ by sampling randomness $x, r \leftarrow \mathbb{Z}_q$ and sets $c = (c_1, c_2, c_3) = (g^x, g^r, m \cdot g^{rx})$. The tuple (x, r) serves as the decommitment information. It is *perfectly* binding and computationally hiding due to the DDH assumption. Given a commitment c to m it can be rerandomized to a commitment $\widehat{c}$ to $\widehat{m} = m \cdot \widetilde{m}$ as follows: Sample $\widetilde{x}, \widetilde{r} \leftarrow_R \mathbb{Z}_q$, compute $\widehat{c} = (\widehat{c_1}, \widehat{c_2}, \widehat{c_3}) = (c_1 \cdot g^{\widetilde{x}}, c_2 \cdot g^{\widetilde{r}}, c_3 \cdot \widetilde{m} \cdot c_1^{\widetilde{r}} \cdot c_2^{\widetilde{x}} \cdot g^{\widetilde{x}\widetilde{r}})$.

3.2 Cryptographic Reverse Firewalls

In this section we recall the basic definitions of reverse firewalls following [MS15, CDN20, CGPS21]. We focus on the setting of two parties.

Notation. Let Π denote a ℓ-round two-party protocol, for some arbitrary polynomial $\ell(\cdot)$ in the security parameter κ. For a party P and reverse firewall RF we define $\mathsf{RF} \circ P$ 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 write $\Pi_{\mathsf{RF}\circ P}$ (resp. $\Pi_{\overline{P}}$) to represent the protocol Π in which the role of a party P is replaced by the composed party $\mathsf{RF} \circ P$ (resp. the tampered implementation $\overline{P}$). We now define the properties that a reverse firewall must satisfy.

Definition 4 (Functionality maintaining). *For any reverse firewall* RF *and a party* P*, let* $\mathsf{RF}^1 \circ P = \mathsf{RF} \circ P$*, and* $\mathsf{RF}^k \circ P = \underbrace{\mathsf{RF} \circ \cdots \circ \mathsf{RF}}_{k \text{ times}} \circ P$*. 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* $\Pi_{\mathsf{RF}^k \circ P}$ *also satisfies* $\mathcal{F}$*, for any polynomially bounded* $k \geq 1$*.*

Definition 5 (Security preservation). *A reverse firewall* strongly preserves security $\mathcal{S}$ for party P in protocol Π *if protocol* Π *satisfies* $\mathcal{S}$*, and for any polynomial-time algorithm* $\overline{P}$*, the protocol* $\Pi_{\mathsf{RF}\circ\overline{P}}$ *satisfies* $\mathcal{S}$*. (i.e., the firewall can guarantee security even when an adversary has tampered with party* P*.)*

A reverse firewall preserves security $\mathcal{S}$ for party P in protocol Π *if protocol* Π *satisfies* $\mathcal{S}$*, and for any polynomial-time algorithm* $\overline{P}$ *such that* $\Pi_{\overline{P}}$ *satisfies* $\mathcal{F}$*, the protocol* $\Pi_{\mathsf{RF}\circ\overline{P}}$ *satisfies* $\mathcal{S}$*. (i.e., the firewall can guarantee security even when an adversary has tampered with* P*, provided that the tampered implementation does not break the functionality of the protocol.)*

We now define exfiltration resistance, which intuitively asks the adversary to distinguish between a tampered implementation $\overline{P}$ of party P from an honest implementation (via the reverse firewall). This prevents, for e.g., for a tampered implementation $\overline{P}$ to leak the secrets of P.

Definition 6 (Exfiltration resistance). *A reverse firewall is* exfiltration resistant for party P_1 against party P_2 in protocol Π *satisfying functionality* $\mathcal{F}$ *if no PPT adversary* $\mathcal{A}_{ER}$ *with output circuits* $\overline{P_1}$ *and* $\overline{P_2}$ *such that* $\Pi_{\overline{P_1}}$ *and* $\Pi_{\overline{P_2}}$ *satisfies* $\mathcal{F}$ *has non-negligible advantage in the game* $\mathsf{LEAK}(\Pi, P_1, P_2, \mathsf{RF}, \kappa)$ *(see Fig.2). If* P_2 *is empty, then we simply say that the firewall is* exfiltration resistant*.*

A reverse firewall is strongly exfiltration resistant for party P_1 against party P_2 in protocol Π *if no PPT adversary* $\mathcal{A}_{\mathsf{ER}}$ *has non-negligible advantage in the game* $\mathsf{LEAK}(\Pi, P_1, P_2, \mathsf{RF}, \kappa)$*. If* P_2 *is empty, then we simply say that the firewall is* strongly exfiltration resistant*.*

$$\frac{\mathsf{LEAK}(\Pi, P_1, P_2, \mathsf{RF}, \kappa)}{(\overline{P_1}, \overline{P_2}, I) \leftarrow \mathcal{A}_{\mathsf{ER}}(1^\kappa)}$$
$$b \xleftarrow{\$} \{0,1\};$$
If $b = 1$, $P_1^* \leftarrow \mathsf{RF}_1 \circ \overline{P_1}$
Else, $P_1^* \leftarrow \mathsf{RF}_1 \circ P_1$.
$\tau^* \leftarrow \Pi_{P_1^*, \{P_2 \to \overline{P_2}\}}(I)$.
$b^* \leftarrow \mathcal{A}_{\mathsf{ER}}(\tau^*, \{\mathsf{st}_{\overline{P_2}}\})$.
Output $(b = b^*)$.

Fig. 2: $\mathsf{LEAK}(\Pi, P_1, P_2, \mathsf{RF}, \kappa)$ is the exfiltration-resistance security game for a reverse firewall RF_1 for party P_1 in the protocol Π against party P_2 with input I. Here, $\mathcal{A}_{\mathsf{ER}}$ is the adversary, $\mathsf{st}_{\overline{P_2}}$ denote the state of party P_2 after the run of the protocol, and τ^* denote the transcript of the protocol $\Pi_{P_1^*, \{P_2 \to \overline{P_2}\}}$ with input I.

We recall the *transparency* property [CGPS21] that intuitively, requires that the behavior of $\mathsf{RF} \circ P$ is identical to the behavior of P if P is the honest implementation.

We will use the following result established in [CGPS21]. It basically states that exfiltration resistance implies security preservation for protocols satisfying simulation-based definition of security.

Theorem 3 ([CGPS21] Exfiltration resistance implies Security preservation). *Let Π denote a two-party protocol running between P_1 and P_2 that securely computes some function f with abort in presence of malicious adversaries in the simulation-based setting. Assume w.l.o.g, that P_1 is honest (i.e., not maliciously corrupted). Then if the reverse firewall RF_1 is* functionality-maintaining*, (strongly/weakly) exfiltration resistant for P_1 against P_2, and* transparent*, then for all PPT adversaries $\mathcal{A}$ and all PPT tempering $\overline{P_1}$ provided by $\mathcal{A}$, the firewall RF_1* (strongly/weakly) preserve security *of the party P_1 in the protocol Π according to Definition 5.*

We recall the definition of transparent firewalls from [CGPS21].

Definition 7 (Transparency). *A reverse firewall is* transparent for an honest party P_1 against party P_2 in protocol Π *if no PPT adversary $\mathcal{A}_{\mathsf{TR}}$ has non-negligible advantage in the game $\mathsf{TRANS}(\Pi, P_1, P_2, \mathsf{RF}_1, \kappa)$ (see Fig.3).*

$\underline{\mathsf{TRANS}(\Pi, P_1, P_2, \mathsf{RF}, \kappa)}$
$I \leftarrow \mathcal{A}_{\mathsf{ER}}(1^\kappa)$
$b \xleftarrow{\$} \{0,1\}$;
If $b = 1$, $P_1^* \leftarrow \mathsf{RF}_1 \circ P_1$
Else, $P_1^* \circ P_1$.
$\tau^* \leftarrow \Pi_{P_1^*, P_2}(I)$.
$b^* \leftarrow \mathcal{A}_{\mathsf{TR}}(\tau^*, \{\mathsf{st}_{P_2}\})$.
Output $(b = b^*)$.

Fig. 3: $\mathsf{TRANS}(\Pi, P_1, P_2, \mathsf{RF}, \kappa)$ is the transparency game for a reverse firewall RF_1 for an honest party P_1 in the protocol Π against party P_2 with input I.

3.3 Zero Knowledge and Witness Indistinguishability

In this section we provide the supplementary material on interactive proofs, sigma protocols, witness indistinguishability and zero knowledge.

Interactive Proofs. Let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$ be an NP relation, with associated language $\mathcal{L}$, *i.e.* $\mathcal{L} = \{x : \exists w \text{ s.t. } (x,w) \in \mathcal{R}\}$. We often call x the statement or theorem, and w the corresponding witness.

An interactive proof system (IPS) for $\mathcal{R}$ is a pair of algorithms $\Pi = (\mathsf{P}, \mathsf{V})$ modeled as interactive PPT Turing machines. The prover algorithm P takes as input a statement $x \in \mathcal{L}$ and a corresponding witness w for x. The verifier algorithm V takes as input a statement x, and at the end of the protocol outputs a decision bit indicating whether it is convinced that $x \in \mathcal{L}$ or not. We write $\mathsf{P}(x,w) \rightleftharpoons \mathsf{V}(x)$ for the random variable corresponding to the view of V in a run of Π on common input x to P, V, and auxiliary input w to P; such view includes the protocol's transcript $\tau \in \{0,1\}^*$ (consisting of all messages exchanged during the protocol) and the internal coin tosses of the verifier. We also write $\langle \mathsf{P}(x,w), \mathsf{V}(x) \rangle$ to denote the random variable corresponding to the decision bit of the verifier in such an execution. An IPS must satisfy the following properties:

- *Completeness.* Let $\Pi = (\mathsf{P}, \mathsf{V})$ be an IPS for a relation $\mathcal{R}$. We say that Π satisfies completeness if for all $(x,w) \in \mathcal{R}$ the following holds:
  $\Pr[\langle \mathsf{P}(x,w), \mathsf{V}(x) \rangle = 1] = 1$.

– *Soundness.* Let $\Pi = (\mathsf{P}, \mathsf{V})$ be an IPS for a relation $\mathcal{R}$. We say that Π satisfies computational soundness if for all $x \notin \mathcal{L}$ and for all PPT malicious provers P^* there exists a negligible function $\nu : \mathsf{N} \to [0, 1]$ such that

$$\Pr[\langle \mathsf{P}^*(x), \mathsf{V}(x) \rangle = 1] \leq \nu(\kappa).$$

– *Zero knowledge.* Let $\Pi = (\mathsf{P}, \mathsf{V})$ be an IPS for a relation $\mathcal{R}$. We say that Π satisfies computational (black-box, auxiliary-input) zero knowledge if there exists a PPT simulator Sim such that for all (non-uniform) PPT malicious verifiers V^* the following holds:

$$\left\{\mathsf{P}(x, w) \rightleftharpoons \mathsf{V}^*(x, z)\right\}_{(x,w)\in\mathcal{R}, z\in\{0,1\}^*} \overset{c}{\approx} \left\{\mathsf{Sim}^{\mathsf{V}^*(x,z,\cdot;\cdot)}(x)\right\}_{x\in\mathcal{L}, z\in\{0,1\}^*},$$

where $\mathsf{V}^*(x, z, \cdot; \cdot)$ denotes the next-message function of the interactive Turing machine V^* when the common input x, and auxiliary input z are fixed.

Witness Indistinguishability (WI). The WI property intuitively says that for any statement $x \in \mathcal{L}$ admitting multiple witnesses w_0, w_1, transcripts produced by having the honest prover use w_0 and w_1 should be computationally indistinguishable, even in case the verifier is malicious. The formal definition follows.

Definition 8 (Witness indistinguishability). *Let $\Pi = (\mathsf{P}, \mathsf{V})$ be an IPS for a relation $\mathcal{R}$. We say that Π satisfies computational (auxiliary-input) witness indistinguishability (WI) if for all (non-uniform) PPT malicious verifiers V^* the following holds:*

$$\left\{\mathsf{P}(x, w_0) \rightleftharpoons \mathsf{V}^*(x, z)\right\}_{(x,w_0)\in\mathcal{R}, z\in\{0,1\}^*} \overset{c}{\approx} \left\{\mathsf{P}(x, w_1) \rightleftharpoons \mathsf{V}^*(x, z)\right\}_{(x,w_1)\in\mathcal{R}, z\in\{0,1\}^*}.$$

In case the above two ensembles are identically distributed, we say that Π satisfies perfect WI.

Zero Knowledge functionality We present the ideal ZK functionality in Fig. 4.

$\mathcal{F}_{\mathsf{ZK}}$
On receiving $(\texttt{prove}, \mathsf{sid}, x, y)$ from P and $(\texttt{verify}, \mathsf{sid}, x')$ from V, output $(\texttt{accept}, x, \mathsf{sid})$ to V if $x == x'$ and $\mathcal{R}(x, y) = 1$, else output $(\texttt{reject}, x, \mathsf{sid})$.

Fig. 4: The Zero-knowledge functionality

Sigma Protocols. Sigma protocols are special class of public-coin interactive proof systems $\Sigma = (\mathsf{P}, \mathsf{V})$ consisting of 3 rounds. The prover speaks first generating a message a, followed by a challenge $c \in \{0, 1\}^\ell$ which is a uniform random string sent by the verifier and then followed by the response z from the prover. The resulting transcript $\tau = (a, c, z)$ is said to be *accepting* w.r.t statement x if $\mathsf{V}(\tau, x) = 1$. Besides completeness, Sigma protocols must satisfy the following properties:

– *Special Soundness.* Let Σ be a Sigma protocol for a relation $\mathcal{R}$. We say that Σ satisfies *special soundness* if there exists a polynomial-time algorithm called the *extractor* which when given x and two transcripts $\tau = (a, c, z)$ and $\tau' = (a, c', z')$ that are accepting for x, with $c \neq c'$, outputs a value w such that $(x, w) \in \mathcal{R}$.

This property is a strong form of soundness which, in fact, implies Sigma protocols are not only sound but even *proofs of knowledge*.

– *Special honest-verifier zero knowledge.* Let Σ be a Sigma protocol for a relation $\mathcal{R}$. We say that Σ satisfies *computational (resp. perfect) special honest-verifier zero knowledge* (SHVZK) if there exists a PPT simulator taking as input x and $c \in \{0,1\}^\ell$, and outputting an accepting transcript for x where c is the challenge, such that the following holds: For all ℓ bit strings c, the distribution of the output of the simulator on input (x, c) is *computationally indistinguishable* from (resp. *identically distributed* to) the distribution of an honest transcript obtained when V sends c as challenge and P runs on common input x and any private input w such that $(x, w) \in \mathcal{R}$.

  Moreover, for our purpose we require the underlying Sigma protocol to be a *unique response* protocol.

– *Unique response.* This states that, there exists no two transcripts $\tau = (a, c, z)$ and $\tau' = (a, c, z')$ with $z \neq z'$ such that $\mathsf{V}(\tau, x) = \mathsf{V}(\tau', x) = 1$. In case that it is only computationally infeasible to find two responses for given x, a, c, the protocol is said to have (computationally) unique responses.

  Most known sigma protocols like knowledge of discrete log, DDH tuple, knowledge representation and other instantiations that can be cast in Maurer's unifying framework are known to be unique response [GMV20].

4 Correlated OT Extension in the Firewall Setting

We describe our revised cOT extension protocol in Fig. 7 and the corresponding firewall can be found in Fig. 8. High level overview can be found in Sec. 2.2. We show security of our protocol by proving Thm. 4.

Theorem 4. *Assuming π_{rOT} implements $\mathcal{F}_{\mathsf{rOT}}$ functionality, Com is a binding and hiding commitment scheme, PRG is a pseudorandom generator and H is a collision resistant hash function, then π_{cOT} implements $\mathcal{F}_{\mathsf{cOT}}$ functionality against active corruption of parties in the $\mathcal{F}_{\mathsf{RO}}$ model.*

Proof. We consider the case for a corrupt sender and the case for a corrupt receiver seperately as follows.

Simulation against a corrupt sender. The simulation algorithm is provided in Fig. 5 and argue indistinguishability between real and ideal world as follows.

– Hyb_0 : Real world execution of the protocol.

– Hyb_1 : Same as Hyb_0, except the simulator invokes the simulator of π_{rOT} to simulate the Seed OT phase. Indistinguishability follows due to simulation based security of π_{rOT} against a corrupt receiver.

– Hyb_2 : Same as Hyb_1, except the simulator performs the coin tossing according to the simulation algorithm, i.e. by running the coin tossing protocol with input $\mathsf{seed}'_{\mathsf{R}}$ and then rewinding the corrupt sender and sending a different $\mathsf{seed}_{\mathsf{R}}$ and aborts if the sender opens to a different share of the seed. An adversary distinguishes between the two hybrids if the simulator aborts in Hyb_2 but does not abort in Hyb_1. This occurs when both $(\mathsf{seed}_{\mathsf{S}}, \delta_{\mathsf{S}})$ and $(\mathsf{seed}'_{\mathsf{S}}, \delta'_{\mathsf{S}})$ are valid openings of the commitment scheme; leading to an adversary breaking binding property of the commitment scheme.

– Hyb_3 : Same as Hyb_2, except the simulator samples $\mathbf{D}$ randomly and computes $\mathbf{v} = \bigoplus_{j \in [m+\kappa]} (\chi_j \cdot \mathbf{R}_j)$, $\mathbf{w}$ using knowledge of $\mathbf{Q}$ and $\mathbf{u} = \mathbf{w} \oplus \mathbf{s} \cdot \mathbf{v}$. The simulator also programs the random oracle on coin. Indistinguishability follows due to PRG security and hiding property of the commitment c_{coin}. The corrupt sender does not possess $k_{i,\overline{s_i}}$ and hence the distribution of $\mathsf{PRG}(k_{i,\overline{s_i}})$ and $\mathbf{D}$ in Hyb_2 is indistinguishable from a random string in Hyb_3. The simulator is able to successfully program the random oracle on coin since the commitment c_{coin} hides an additive share, i.e. c_{R}, of coin.

– Hyb_4 : Same as Hyb_3, except the simulator computes $\mathbf{v}$ and $\mathbf{u}$ according to simulation algorithm. This is the ideal world execution of the protocol. $(\mathbf{R}_{m+1}, \ldots, \mathbf{R}_{m+\kappa})$ are chosen uniformly at random in Hyb_3 and contains κ bits of entropy. $(\mathbf{R}_{m+1}, \ldots, \mathbf{R}_{m+\kappa})$ is hidden from the corrupt sender and as a result it statistically hides $\mathbf{r}$ in $\mathbf{v}$.

Seed OT Phase:

1. The simulator invokes the simulator of π_{rOT} to obtain the corrupt sender's output from π_{rOT} as $(\mathbf{s}, \mathbf{k}')$.

OT Extension Phase:

1. The simulated receiver performs the coin-tossing protocol honestly to obtain coin, commitment to coin_R and the decommitment δ_{coin}.
2. The simulated receiver samples a random $\mathbf{D}^1 \leftarrow_R \mathbb{F}^{(m+\kappa)\times\kappa}$ and commits to it as $c_\mathbf{D} = \mathsf{Com}(H(\mathbf{D}); \delta_\mathbf{D})$ using randomness d and sends $c_\mathbf{D}$ to S. The simulated receiver also programs the random oracle $\mathcal{F}_{\mathsf{RO}}$ on coin such that $\mathbf{D}^1 = \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, \mathsf{coin})$.

Consistency Check Phase:

1. The corrupt sender and the simulated receiver performs the coin tossing protocol as follows:
   - The corrupt sender sends commitment c_{seed} to the simulated receiver.
   - The simulated receiver samples $\mathsf{seed}'_\mathsf{R} \leftarrow_R \{0,1\}^\kappa$ and sends $\mathsf{seed}'_\mathsf{R}$ to S.
   - The corrupt sender opens c_{seed} by sending $(\mathsf{seed}'_\mathsf{S}, \delta'_{\mathsf{seed}})$ to the simulator.
   - The simulator rewinds the sender and sends $\mathsf{seed}_\mathsf{R} \leftarrow_R \{0,1\}^\kappa$ to the sender.
   - After rewinding, the corrupt sender opens c_{seed} by sending $(\mathsf{seed}'_\mathsf{S}, \delta'_{\mathsf{seed}})$ to the simulated receiver. If $(\mathsf{seed}'_\mathsf{S}, \delta'_{\mathsf{seed}}) \neq (\mathsf{seed}_\mathsf{S}, \delta_{\mathsf{seed}})$ then the simulator aborts.
2. The simulator aborts if $c_{\mathsf{seed}} \neq \mathsf{Com}(\mathsf{seed}_\mathsf{S}; \delta_{\mathsf{seed}})$. Else, the simulator computes challenge from the output of the coin tossing protocol, as $\chi = \{\chi_1, \ldots, \chi_{m+\kappa}\} = \mathsf{PRG}(\mathsf{seed}_\mathsf{S} + \mathsf{seed}_\mathsf{R})$.
3. The simulator computes $\mathbf{Q}$ such that $\mathbf{Q}^i = \left(s_i \odot \mathbf{D}^i\right) \oplus \mathsf{PRG}(k'_i)$.
4. The simulator samples $r \leftarrow_R \{0,1\}$, $\alpha \leftarrow_R \mathbb{F}$ and sets $\mathbf{v} = \alpha \cdot (r, \ldots, r) \in \mathbb{F}^\kappa$. The simulator computes $\mathbf{w} = \bigoplus_{j\in[m+\kappa]}(\chi_j \cdot \mathbf{Q}_j)$ where $\chi = \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, \mathsf{seed})$. The simulator sets $\mathbf{u} = \mathbf{w} \oplus \mathbf{s} \cdot \mathbf{v}$. The simulator sends $(\mathbf{D}, \delta_\mathbf{D}, \mathbf{u}, \mathbf{v})$ to the corrupt sender. The simulator also decommits to coin_R by sending $(\mathsf{coin}_\mathsf{R}, \delta_{\mathsf{coin}})$ to the corrupt sender.

Output Phase:
The simulator invokes $\mathcal{F}_{\mathsf{cOT}}$ with the corrupt sender's input as $(\mathbf{s}, \mathbf{Q})$.

Fig. 5: Simulation against a corrupt sender in π_{cOT}

Simulation against a corrupt receiver. The simulation algorithm is provided in Fig. 6 and argue indistinguishability between real and ideal world as follows.

- Hyb_0 : Real world execution of the protocol.
- Hyb_1 : Same as Hyb_0, except the simulator invokes the simulator of π_{rOT} to obtain the corrupt receiver's output from π_{rOT} as $(\mathbf{k}_0, \mathbf{k}_1)$. Indistinguishability follows due to security of π_{rOT} against a corrupt sender.

- Hyb_2 : Same as Hyb_1, except the simulator performs the first rewinding to the OT extension phase, programs $\mathcal{F}_{\mathsf{RO}}$ such that it opens to a random $\mathbf{D}^1$ and the simulator aborts if the corrupt receiver opens c_{coin} to two different openings $(\mathsf{coin}''_\mathsf{R}, \delta''_{\mathsf{coin}}) \neq (\mathsf{coin}'_\mathsf{R}, \delta'_{\mathsf{coin}})$. Indistinguishability follows due to binding of the commitment scheme for c_{coin} and $\mathbf{D}^1$ looks random (on the uniformly sampled unqueried point $\mathsf{coin}_r ec'' \oplus \mathsf{coin}_\mathsf{S}$) due to the random oracle assumption.
- Hyb_3 : Same as Hyb_2, except the simulator performs the second rewinding after the coin tossing protocol and reruns the coin tossing protocol and aborts if the coin tossing outputs are same, i.e. $(\mathsf{seed}'_\mathsf{S} + \mathsf{seed}'_\mathsf{R}) == (\mathsf{seed}_\mathsf{S} + \mathsf{seed}_\mathsf{R})$. This occurs when the adversary breaks the hiding of the commitment scheme Com.
- Hyb_4 : Same as Hyb_3, except the simulator rewinds the receiver after the coin tossing protocol and reruns the coin tossing protocol and aborts if the corrupt receiver opened $c_\mathbf{D}$ to two different openings $(\mathbf{D}', \delta'_\mathbf{D}) \neq (\mathbf{D}, \delta_\mathbf{D})$ or opened c_{coin} to two different openings $(\mathsf{coin}_\mathsf{R}, \delta_\mathsf{R}) \neq (\mathsf{coin}''_\mathsf{R}, \delta''_\mathsf{R})$. This occurs when either the adversary breaks the binding of the commitment scheme Com with openings $(H(\mathbf{D}'), \delta'_\mathbf{D})$ and $(H(\mathbf{D}), \delta_\mathbf{D})$, or it finds a collision in the hash function H such that $H(\mathbf{D}') == H(\mathbf{D})$ for $\mathbf{D} \neq \mathbf{D}'$, or it breaks the binding property of c_{coin} with openings $(\mathsf{coin}_\mathsf{R}, \delta_\mathsf{R}) \neq (\mathsf{coin}''_\mathsf{R}, \delta''_\mathsf{R})$.

Seed OT Phase:

1. The simulator invokes the simulator of π_{rOT} to obtain the corrupt receiver's output from π_{rOT} as $(\mathbf{k}_0, \mathbf{k}_1)$.

OT Extension Phase and Consistency Check:

1. The simulated sender runs the coin tossing protocol honestly to obtain c_{coin} from the corrupt receiver. It also receives $c_{\mathbf{D}}$ from the receiver.
2. The simulator runs the coin tossing honestly to generate c_{seed} and seed''.
3. The corrupt receiver sends $(\mathbf{D}'', \delta''_{\mathbf{D}}, \mathbf{u}'', \mathbf{v}'')$ and $(\mathsf{coin}''_{\mathsf{R}}, \delta''_{\mathsf{coin}})$. The simulator aborts if $c_{\mathsf{coin}} \neq \mathsf{Com}(\mathsf{coin}''_{\mathsf{R}}; \delta''_{\mathsf{coin}})$.
4. *First Rewinding:* The simulator rewinds the receiver to Step 1 of the OT extension phase and samples a different $\mathsf{coin}_{\mathsf{S}}$ after obtaining c_{seed} from R. The simulator samples a $\mathbf{D}^1 \leftarrow_R \mathbb{F}^{m+\kappa}$ and programs the $\mathcal{F}_{\mathsf{RO}}$ such that $\mathbf{D}^1 = \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, \mathsf{coin}''_{\mathsf{R}} \oplus \mathsf{coin}_{\mathsf{S}})$. The simulator sends $\mathsf{coin}_{\mathsf{S}}$ to the receiver as part of the coin tossing subprotocol. The corrupt R sends $c_{\mathbf{D}}$ to the sender.
5. The simulator runs the coin tossing protocol honestly with the corrupt receiver as follows:
   - The simulator samples $\mathsf{seed}'_{\mathsf{S}} \leftarrow_R \{0,1\}^\kappa$ and sends $c'_{\mathsf{seed}} = \mathsf{Com}(\mathsf{seed}'_{\mathsf{S}}; \delta'_{\mathsf{seed}})$ to the corrupt receiver.
   - The corrupt receiver sends $\mathsf{seed}'_{\mathsf{R}}$ to the simulated sender.
   - The simulator opens c'_{seed} by sending $(\mathsf{seed}'_{\mathsf{S}}, \delta'_{\mathsf{seed}})$ to the corrupt receiver.
6. The corrupt receiver sends $(\mathbf{D}', \delta'_{\mathbf{D}}, \mathbf{u}', \mathbf{v}')$ and $(\mathsf{coin}'_{\mathsf{R}}, \delta'_{\mathsf{R}})$ to S as the response. If $(\mathsf{coin}'_{\mathsf{R}}, \delta'_{\mathsf{R}}) \neq (\mathsf{coin}''_{\mathsf{R}}, \delta''_{\mathsf{R}})$ then the simulator aborts.
7. *Second Rewinding:* The simulator rewinds the receiver to Step 1 of Consistency Check Phase and runs the coin-tossing protocol with different randomness as follows:
   - The simulator samples $\mathsf{seed}_{\mathsf{S}} \leftarrow_R \{0,1\}^\kappa$ and sends $c_{\mathsf{seed}} = \mathsf{Com}(\mathsf{seed}_{\mathsf{S}}; \delta_{\mathsf{seed}})$ to the corrupt receiver.
   - The corrupt receiver sends $\mathsf{seed}_{\mathsf{R}}$ to the simulated sender.
   - The simulator opens c_{seed} by sending $(\mathsf{seed}_{\mathsf{S}}, \delta_{\mathsf{seed}})$ to the corrupt receiver.
8. The corrupt receiver sends $(\mathbf{D}, \delta_{\mathbf{D}}, \mathbf{u}, \mathbf{v})$ and $(\mathsf{coin}_{\mathsf{R}}, \delta_{\mathsf{R}})$ to S as the response.
9. Upon receiving the response, the simulator aborts if any of the following occurs:
   - The coin tossing outputs are same - $(\mathsf{seed}'_{\mathsf{S}} + \mathsf{seed}'_{\mathsf{R}}) == (\mathsf{seed}_{\mathsf{S}} + \mathsf{seed}_{\mathsf{R}})$.
   - The corrupt receiver opened $c_{\mathbf{D}}$ to two different openings $(\mathbf{D}', \delta'_{\mathbf{D}}) \neq (\mathbf{D}, \delta_{\mathbf{D}})$.
   - The corrupt receiver opened c_{coin} to two different openings, i.e. $(\mathsf{coin}_{\mathsf{R}}, \delta_{\mathsf{R}}) \neq (\mathsf{coin}''_{\mathsf{R}}, \delta''_{\mathsf{R}})$.
10. The simulator constructs matrix $\mathbf{R}$ column wise as follows for $i \in [\kappa]$:
    - Computes $\mathbf{M}^i = \mathsf{PRG}(k_{i,0})$.
    - Computes $\mathbf{R}^i = \mathbf{M}^i \oplus \mathsf{PRG}(k_{i,1}) \oplus \mathbf{D}^i$.
11. The simulator denotes $\mathbf{R}$ columnwise as $\mathbf{R} = (\mathbf{R}^1, \ldots, \mathbf{R}^\kappa)$. The simulator aborts if μ columns differ. Else, set $\mathbf{r}$ as the column which matches with μ columns of $\mathbf{R}$.
12. The simulator sets $\mathbf{ind} = (i_1, i_2, \ldots, i_c)$ as the set of columns of $\mathbf{R}$ which differ from $\mathbf{r}$. It sets $\mathbf{s}' = 0^c$. The simulator invokes the $\mathcal{F}_{\mathsf{rOT}}$ functionality with input $(\textsc{Guess}, \mathsf{sid}, \mathbf{ind}, \mathbf{s}')$. If $\mathcal{F}_{\mathsf{rOT}}$ aborts then the simulator also aborts and returns the view of the dummy adversary as the ideal world adversary view. Else, if $\mathcal{F}_{\mathsf{rOT}}$ returns $(\textsc{Undetected}, \mathsf{sid})$ then the simulator proceeds with the simulation.

Output Phase:
The simulator invokes $\mathcal{F}_{\mathsf{cOT}}$ with the corrupt receiver's input as $(\mathbf{r}, \mathbf{M})$.

Fig. 6: Simulation against a corrupt receiver in π_{cOT}

– Hyb_5 : Same as Hyb_4, except the simulator constructs $\mathbf{M}$ and $\mathbf{R}$ matrix following the simulation algorithm. The simulator samples $\mathbf{s} \leftarrow_R \{0,1\}^\kappa$ and computes $\mathbf{Q}$ and $\mathbf{w}$, honestly performs the consistency check and the simulator aborts if μ columns of $\mathbf{R}$ differ. If abort has not occurred then the simulator parses computes $\mathbf{r}$ as per simulation algorithm and invokes $\mathcal{F}_{\mathsf{cOT}}$ with $\mathbf{r}$ and $\mathbf{M}$ as the corrupt receiver's input. The adversary distinguishes between the two hybrids if it passes the consistency check and correctly guesses μ bits of $\mathbf{s}$. In such a case, Hyb_5 results in an abort; whereas Hyb_4 does not. This occurs with $2^{-\mu}$ negligible probability [Dia22] in the random oracle model since χ is ensured to be random on the random chosen point seed. This is the ideal world execution of the protocol.

□

– **Private Inputs:** R and S do not possess any private inputs.
– **Primitives:** Pseudorandom Generators $\mathsf{PRG} : \{0,1\}^\kappa \to \{0,1\}^{m+\kappa}$ and $\mathcal{F}_{\mathsf{RO}} : \{0,1\}^\kappa \to \mathbb{F}^{m+\kappa}$ is a random oracle functionality and Collision Resistant Hash function $H : \{0,1\}^{(m+\kappa)\times\kappa} \to \{0,1\}^\kappa$ and $\mathsf{Com} : \{0,1\}^\kappa \leftarrow \{0,1\}^\kappa$ is a string commitment scheme.
– **Subprotocols:** Subprotocol π_{rOT} computes ℓ instance of random OT.

Seed OT Phase:

1. S and R participate in π_{rOT} protocol (implementing the $\mathcal{F}_{\mathsf{rOT}}$ functionality) as receiver and sender respectively.
2. R receives $(\mathbf{k}_0, \mathbf{k}_1)$ as output where $\mathbf{k}_\alpha = \{k_{i,\alpha}\}_{i\in[\kappa]}$ and $k_{i,\alpha} \in \{0,1\}^\kappa$ for $\alpha \in \{0,1\}, i \in [\kappa]$.
3. S receives $\mathbf{s} \in \{0,1\}^\kappa$ and $\mathbf{k}'$ where $\mathbf{k}' = \{k'_i\}_{i\in[\kappa]}$ and $k'_i = k_{i,s_i}$ for $i \in [\kappa]$.

OT Extension Phase:

1. R and S perform a coin tossing protocol as follows:
   – R samples $\mathsf{coin}_\mathsf{R} \leftarrow_R \{0,1\}^\kappa$ and sends $c_{\mathsf{coin}} = \mathsf{Com}(\mathsf{coin}_\mathsf{R}; \delta_{\mathsf{coin}})$ to S.
   – S obtains c_{coin} and samples $\mathsf{coin}_\mathsf{S} \leftarrow_R \{0,1\}^\kappa$ and sends coin_S to R.
   – R computes $\mathsf{coin} = \mathsf{coin}_\mathsf{R} \oplus \mathsf{coin}_\mathsf{S}$.
2. R forms three $(m+\kappa)\times\kappa$ matrices $\mathbf{M}$, $\mathbf{R}$ and $\mathbf{D}$ in the following way:
   – Sets $\mathbf{M}^i = \mathsf{PRG}(k_{i,0})$ for $i \in [\kappa]$.
   – Sets $\mathbf{D}^1 = \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, \mathsf{coin})$. Computes $\mathbf{r}' = \mathbf{D}^1 \oplus \mathbf{M}^1 \oplus \mathsf{PRG}(k_{1,1})$.
   – Parses $\mathbf{r}' = \mathbf{r}||\tau$ where $\mathbf{r} \in \{0,1\}^m$ and $\tau \in \{0,1\}^\kappa$.
   – Sets $\mathbf{R}_j = (r'_j, \ldots, r'_j)$ for $j \in [m+\kappa]$. Clearly, $\mathbf{R}^i = \mathbf{r}'$ for $i \in [\kappa]$.
   – Set $\mathbf{D}^i = \mathbf{M}^i \oplus \mathsf{PRG}(k_{i,1}) \oplus \mathbf{R}^i$ for $i \in [\kappa]$.
   R sets $\mathbf{D} = \{\mathbf{D}^i\}_{i\in[\kappa]}$. R commits to $\mathbf{D}$ as $c_\mathbf{D} = \mathsf{Com}(H(\mathbf{D}); \delta_\mathbf{D})$ using randomness d and sends $c_\mathbf{D}$ to S.

Consistency Check Phase:

1. S and R performs a coin tossing protocol as follows:
   – S samples $\mathsf{seed}_\mathsf{S} \leftarrow_R \{0,1\}^\kappa$ and sends $c_{\mathsf{seed}} = \mathsf{Com}(\mathsf{seed}_\mathsf{S}; \delta_{\mathsf{seed}})$ to R.
   – R obtains c_{seed} and samples $\mathsf{seed}_\mathsf{R} \leftarrow_R \{0,1\}^\kappa$ and sends seed_R to S.
   – S opens c_{seed} by sending $(\mathsf{seed}_\mathsf{S}, \delta_{\mathsf{seed}})$ to R and sets $\mathsf{seed} = \mathsf{seed}_\mathsf{S} + \mathsf{seed}_\mathsf{R}$.
2. R aborts if $c_{\mathsf{seed}} \neq \mathsf{Com}(\mathsf{seed}_\mathsf{S}; \delta_{\mathsf{seed}})$. Else R computes challenge from the output of the coin tossing protocol, as $\chi = \{\chi_1, \ldots, \chi_{m+\kappa}\} = \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, \mathsf{seed}_\mathsf{S} + \mathsf{seed}_\mathsf{R})$.
3. R computes $\mathbf{u} = \bigoplus_{j\in(m+\kappa)}(\chi_j \cdot \mathbf{M}_j)$ and $\mathbf{v} = \bigoplus_{j\in(m+\kappa)}(\chi_j \cdot \mathbf{R}_j)$. R sends $(\mathbf{D}, \delta_\mathbf{D}, \mathbf{u}, \mathbf{v})$ to S as the response. R also decommits c_{coin} to coin_R by sending $(\mathsf{coin}_\mathsf{R}, \delta_{\mathsf{coin}})$.
4. On receiving $\mathbf{D}$, S aborts if $c_\mathbf{D} \neq \mathsf{Com}(H(\mathbf{D}); \delta_\mathbf{D})$ or $c_{\mathsf{coin}} \neq \mathsf{Com}(\mathsf{coin}_\mathsf{R}; \delta_{\mathsf{coin}})$ or $\mathbf{D}^1 \neq \mathcal{F}_{\mathsf{RO}}(\mathsf{coin}_\mathsf{R} \oplus \mathsf{coin}_\mathsf{S})$. S forms $(m+\kappa)\times\kappa$ bit-matrix $\mathbf{Q}$ with the ith column of $\mathbf{Q}$ set as $\mathbf{Q}^i = \big(s_i \odot \mathbf{D}^i\big) \oplus \mathsf{PRG}(k'_i)$. Clearly, (i) $\mathbf{Q}^i = \big(\mathbf{M}^i \oplus (s_i \odot \mathbf{R}^i)\big)$ and (ii) $\mathbf{Q}_j = \big(\mathbf{M}_j \oplus (\mathbf{s} \odot \mathbf{R}_j)\big) = \big(\mathbf{M}_j \oplus (\mathbf{s} \odot r_j)\big)$.
5. S constructs $\chi = \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, \mathsf{seed})$ and computes $\mathbf{w} = \bigoplus_{j\in(m+\kappa)}(\chi_j \cdot \mathbf{Q}_j)$. S aborts if $\mathbf{w} \neq \mathbf{u} \oplus \mathbf{s} \cdot \mathbf{v}$.

Output Phase:
S sets $(\mathbf{s}, \mathbf{Q})$ as the output. R sets $(\mathbf{r}, \mathbf{M})$ as the output.

Fig. 7: Correlated OT Extension π_{cOT} in the RF setting

Com is an additively homomorphic commitment where $\mathsf{Com}(m_1; r_1) \cdot \mathsf{Com}(m_2; r_2) = \mathsf{Com}(m_1 + m_2; r_1 + r_2)$.

Seed OT Phase:
$\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ (resp. $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$) invokes the firewall $\mathsf{RF}_{\mathsf{rOT\text{-}R}}$ (resp. $\mathsf{RF}_{\mathsf{rOT\text{-}S}}$) of base-OT receiver (resp. sender) for sanitising the cOT-extension sender's (resp. receiver's) π_{rOT} messages.

OT Extension Phase:

1. The firewall sanitizes the coin-tossing protocol as follows:
   - Upon receiving c_{coin} from R the firewall samples $\widehat{c_{\mathsf{coin}}} = c_{\mathsf{coin}} \cdot \mathsf{Com}(\widetilde{\mathsf{coin}}; \widetilde{\delta_{\mathsf{coin}}})$ where $\widetilde{\mathsf{coin}} \leftarrow_R \{0,1\}^*$ and $\widetilde{\delta_{\mathsf{coin}}} \leftarrow_R \{0,1\}^*$. The firewall sends $\widehat{c_{\mathsf{coin}}}$ to the sender.
   - Upon receiving $\mathsf{coin_S}$ from the sender, the firewall sends $\widehat{\mathsf{coin_S}} = \mathsf{coin_S} + \widetilde{\mathsf{coin}}$ to the receiver.
2. Upon receiving $c_{\mathbf{D}}$ from receiver, the firewall computes $\widehat{c_{\mathbf{D}}} = c_{\mathbf{D}} \cdot \mathsf{Com}(0; \widetilde{\delta_{\mathbf{D}}})$ where $\widetilde{\delta_{\mathbf{D}}} \leftarrow_R \{0,1\}^*$. The firewall sends $\widehat{c_{\mathbf{D}}}$ to the receiver.

Consistency Check Phase:

1. The firewall sanitizes the coin tossing protocol messages as follows:
   - When S sends c_{seed}, the firewall samples $\widetilde{\mathsf{seed}}$ and computes the sanitized commitment as $\widehat{c_{\mathsf{seed}}} = c_{\mathsf{seed}} \cdot \mathsf{Com}(\widetilde{\mathsf{seed}}; \widetilde{\delta_{\mathsf{seed}}})$ where $\widetilde{\delta_{\mathsf{seed}}} \leftarrow_R \{0,1\}^*$. the firewall sends $\widehat{c_{\mathsf{seed}}}$ to the receiver R.
   - When R sends $\mathsf{seed_R}$, the firewall sends $\widehat{\mathsf{seed_R}} = \mathsf{seed_R} + \widetilde{\mathsf{seed}}$ to the sender S.
   - When S sends $(\mathsf{seed_S}, \delta_{\mathsf{seed}})$, the firewall sends $(\widehat{\mathsf{seed_S}}, \widehat{\delta_{\mathsf{seed}}}) = (\mathsf{seed_S} + \widetilde{\mathsf{seed}}, \delta_{\mathsf{seed}} + \widetilde{\delta_{\mathsf{seed}}})$ to the receiver R.
2. When R sends $(\mathbf{D}, \delta_{\mathbf{D}}, \mathbf{u}, \mathbf{v})$, the firewall computes $\widehat{\delta_{\mathbf{D}}} = \delta_{\mathbf{D}} + \widetilde{\delta_{\mathbf{D}}}$ and sends $(\mathbf{D}, \widehat{\delta_{\mathbf{D}}}, \mathbf{u}, \mathbf{v})$ to S. When R sends $(\mathsf{coin_R}, \delta_{\mathsf{coin}})$, the firewall sends $(\mathsf{coin_R} + \widetilde{\mathsf{coin}}, \delta_{\mathsf{coin}} + \widetilde{\delta_{\mathsf{coin}}})$ to the sender.

Fig. 8: Sender's (resp. Receiver's) Firewall $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ (resp. $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$) in π_{cOT}

Theorem 5. *Assuming* Com *is an additively homomorphic, binding and hiding commitment scheme, and* $\mathsf{RF}_{\mathsf{rOT\text{-}R}}$ *provides exfiltration resistance for the receiver (of base OT) in* π_{rOT} *then* $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ *(Fig. 8) provides exfiltration resistance for a tampered sender of* π_{cOT}*. Similarly if* $\mathsf{RF}_{\mathsf{rOT\text{-}S}}$ *provides exfiltration resistance for the sender (of base OT) in* π_{rOT} *then* $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$ *(Fig. 8) provides exfiltration resistance for a tampered receiver in* π_{cOT}*.*

Proof. We argue exfiltration resistance for each phase as follows:

- The $\mathsf{RF}_{\mathsf{rOT}}$ transcript provides ER to the sender and receiver due to ER of $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$ and $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ respectively.
- In the OT extension phase, the $\widehat{c_{\mathsf{coin}}}$ and $\widehat{c_{\mathbf{D}}}$ provides ER due to homomorphism and hiding property of the commitment scheme.
- In the consistency check phase if a receiver passes the consistency check the random oracle $\mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, \mathsf{coin})$, $\mathsf{PRG}(k_{1,0})$ and $\mathsf{PRG}(k_{1,1})$ ensures that the first column of $\mathbf{D}$ is randomly distributed and as a result $\mathbf{r}'$ is random. Both parties generate the sanitized $\mathbf{r}'$ as follows:

$$\mathbf{r}' = \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, \mathsf{coin_R} + \mathsf{coin_S} + \widetilde{\mathsf{coin}}) \oplus \mathsf{PRG}(k_{1,0}) \oplus \mathsf{PRG}(k_{1,1}),$$

  where $k_{1,0}$ and $k_{1,1}$ are outputs from the sanitized base OT protocols. $\widehat{c_{\mathsf{seed}}}$ provides ER due to to homomorphism and hiding property of the commitment scheme. The consistency check ensures that a malformed $\mathbf{D}$ is detected. For example if the ith column of $\mathbf{D}$ is malformed such that $\mathbf{R}^i \neq \mathbf{r}$ then the check detects and the honest and tampered party aborts when $s_i == 1$. When $s_i == 0$ the check fails to detect it and the adversary is able to leak the ith bit of $\mathbf{s}$. The honest sender does not abort following the protocol and the tampered sender also doesn't abort since it is functionality maintaining w.r.t $\mathcal{F}_{\mathsf{cOT}}$ which enables adversary to guess c bits of $\mathbf{s}$.

□

By composing Theorems 3, 5 and 4 we show that the firewalls $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$ and $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ (Fig. 8) preserves the security of the underlying protocol π_{cOT} and that proves Thm. 1.

5 Implementing $\mathcal{F}_{\mathsf{rOT}}$ in the Firewall Setting

In this section we implement $\mathcal{F}_{\mathsf{rOT}}$ (Fig. 9) for base OT protocol. Our protocol π_{rOT} can be found in Fig. 10 and 11. Detailed overview can be found in Sec. 2.3. We show simulation based security of π_{rOT} by proving Thm. 6. We implement the ZK protocol in Fig. 16 and WI protocol in Fig. 19 in Sec. 6.

Functionality $\mathcal{F}_{\mathsf{rOT}}$

Upon receiving $(\textsc{Initiate}, \mathsf{sid}, \ell)$ from sender S and a receiver R, the functionality $\mathcal{F}_{\mathsf{rOT}}$ interacts as follows:

- If S is corrupted receive $(\mathbf{a}_0, \mathbf{a}_1) \in \{0,1\}^{\ell \times \kappa}$ from the sender. Else, sample $a_{i,0}, a_{i,1} \leftarrow_R \{0,1\}^{\kappa}$ for $i \in [\ell]$ and set $(\mathbf{a}_0, \mathbf{a}_1) = \{a_{i,0}, a_{i,1}\}_{i \in [\ell]}$.
- If R is corrupted then receive $\mathbf{b} \in \{0,1\}^{\ell}$ and $\mathbf{a}' \in \{0,1\}^{\ell \times \kappa}$ from the receiver, and set $a_{i,b_i} = a'_i$ for $i \in [\ell]$. Else, sample $\mathbf{b} \leftarrow_R \{0,1\}^{\ell}$.
- Denote $\mathbf{b} = \{b_i\}_{i \in [\ell]}$. Set $\mathbf{a}' = \{a'_i\}_{i \in [\ell]}$ where $a'_i = a_{i,b_i}$ for $i \in [\ell]$.

Send $(\mathsf{sent}, \mathsf{sid}, (\mathbf{b}, \mathbf{a}'))$ to R and $(\mathsf{sent}, \mathsf{sid}, (\mathbf{a}_0, \mathbf{a}_1))$ to S and store $(\mathsf{sen}, \mathsf{sid}, \ell, (\mathbf{b}, \mathbf{a}_0, \mathbf{a}_1))$ in memory. Ignore future messages with the same sid.

Fig. 9: The ideal functionality $\mathcal{F}_{\mathsf{rOT}}$ for Oblivious Transfer with random inputs

Theorem 6. *Assuming $\mathsf{Com}_{\mathbb{G}}$ and Com_q be computationally binding and hiding commitment schemes where they are rerandomizable and additively homomorphic for message spaces over $\mathbb{G}$ and $\mathbb{Z}_q$ elements respectively, $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ implement $\mathcal{F}_{\mathsf{ZK}}$ functionality for the Discrete Log relation $\mathcal{R}_{\mathsf{DL}}$, $\pi^{\mathsf{OR}}_{\mathsf{WI}}$ be a protocol for Witness Indistinguishability with proof of knowledge for the relation $\mathcal{R}_{\mathsf{OR}}$ and DDH assumption holds in group $\mathbb{G}$, then π_{rOT} implements $\mathcal{F}_{\mathsf{rOT}}$ against active corruption of parties.*

Proof. We consider the case for a corrupt sender and the case for a corrupt receiver seperately as follows.

Simulation against a corrupt sender. The simulation algorithm is provided in Fig. 12 and argue indistinguishability between real and ideal world as follows.

- Hyb_0 : Real world execution of the protocol.
- Hyb_1 : Same as Hyb_0, except the simulator invokes the simulator of $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ to either correctly extract sender's witness q or abort the protocol if extraction fails. Indistinguishability follows from simulation based security against a corrupt prover of $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$. Proof of knowledge property of $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ ensures the existence of a witness extractor if the proof is accepting.
- Hyb_2 : Same as Hyb_1, except the simulator samples $\mathsf{sk}_{i,0} \leftarrow_R \mathbb{Z}_q$, computes $\mathsf{pk}_{i,0} = g^{\mathsf{sk}_{i,0}}$ and runs the ith $\pi^{\mathsf{OR}}_{\mathsf{WI}}$ protocol with witness $(\mathsf{sk}_{i,0}, 0)$ for $i \in [\ell]$. Indistinguishability follows from the WI property of $\pi^{\mathsf{OR}}_{\mathsf{WI}}$. The simulator also computes $\mathsf{sk}_{i,1} = q - \mathsf{sk}_{i,0}$ and the honestly sampled and the simulated public keys are identically distributed in Step 4. This is the ideal world execution of the protocol. The simulated receiver correctly extracts the sender's inputs as $(\mathbf{k}_0, \mathbf{k}_1)$ and invokes $\mathcal{F}_{\mathsf{rOT}}$ functionality with the extracted inputs, following the simulation algorithm.

Simulation against a corrupt receiver. The simulation algorithm is provided in Fig. 13 and argue indistinguishability between real and ideal world as follows.

- Hyb_0 : Real world execution of the protocol.

$\mathsf{Com}_{\mathbb{G}}$ and Com_q are commitments for group elements and field elements respectively. $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ is a ZK proof for the statement $(x, \mathbb{G}, \mathbb{Z}_q)$ corresponding to relation $\mathcal{R}_{\mathsf{DL}} = (\exists w \in \mathbb{Z}_q : x = g^w)$. $\pi^{\mathsf{OR}}_{\mathsf{WI}}$ is a WI proof for the statement $(x_0, x_1, \mathbb{G}, \mathbb{Z}_q)$ corresponding to relation $\mathcal{R}_{\mathsf{OR}} = (\exists w \in \mathbb{Z}_q, b \in \{0,1\} : x_0 = g^w \vee x_1 = g^w)$.

---

1. **Receiver's Coin-tossing for Parameters:** The receiver samples $Q_{\mathsf{R}} \leftarrow_R \mathbb{G}$ and sends $T = \mathsf{Com}_{\mathbb{G}}(Q_{\mathsf{R}}; t)$ to the sender.
2. **Sender's Coin-tossing for Parameters and Receiver's Public Key:** The sender samples $q \leftarrow_R \mathbb{Z}_q$ and computes $Q_{\mathsf{S}} = g^q$. For $i \in [\ell]$ the sender performs the following:
   - The sender samples $p_i \leftarrow_R \mathbb{Z}_q$ to rerandomize the receiver public key.
   - The sender computes $c^{\mathsf{S}}_i = \mathsf{Com}_q(p_i; d^{\mathsf{S}}_i)$.

   The sender sends $(Q_{\mathsf{S}}, \mathbf{C}^{\mathsf{S}})$ to the receiver, where $\mathbf{C}^{\mathsf{S}} = \{c^{\mathsf{S}}_i\}_{i\in[\ell]}$.
3. **Sender's Zero-Knowledge Proof for Parameters:** The sender and the receiver run $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ protocol where sender is the prover for the statement $(Q_{\mathsf{S}}, \mathbb{G}, \mathbb{Z}_q)$ corresponding to witness q.
4. **Receiver's generates Public Keys and Performs Coin-tossing for Sender's OT message:** The receiver computes $Q = Q_{\mathsf{R}} \cdot Q_{\mathsf{S}}$. The receiver samples random choice bits $\mathbf{b} \leftarrow_R \{0,1\}^\ell$. For $i \in [\ell]$ the receiver performs the following:
   - The receiver samples $\mathsf{sk}_i \leftarrow_R \mathbb{Z}_q$ and computes $\mathsf{pk}_{i,b} = g^{\mathsf{sk}_i}$.
   - The receiver computes $\mathsf{pk}_{i,\bar{b}} = \frac{Q}{\mathsf{pk}_{i,b}}$.
   - The receiver samples shares for sender's OT randomness $v_{i,0}, v_{i,1} \leftarrow_R \mathbb{Z}_q$.
   - The receiver commits to the shares as $c^{\mathsf{R}}_{i,0} = \mathsf{Com}_q(v_{i,0}; d^{\mathsf{R}}_{i,0})$ and $c^{\mathsf{R}}_{i,1} = \mathsf{Com}_q(v_{i,1}; d^{\mathsf{R}}_{i,1})$.

   The receiver decommits to T by sending (Q_{R}, t). The receiver also sends the commitments - $(\mathbf{C}^{\mathsf{R}}_0, \mathbf{C}^{\mathsf{R}}_1)$ where $\mathbf{C}^{\mathsf{R}}_0 = \{c^{\mathsf{R}}_{i,0}\}_{i\in[\ell]}$ and $\mathbf{C}^{\mathsf{R}}_1 = \{c^{\mathsf{R}}_{i,1}\}_{i\in[\ell]}$ and the public keys $\{\mathsf{pk}_{i,0}\}_{i\in[\ell]}$.
5. **Receiver's WI Proof for Secret Keys:** For $i \in [\ell]$, the receiver and the sender parallely run $\pi^{\mathsf{OR}}_{\mathsf{WI}}$ protocol where receiver is the prover for the statement $\{\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1}, \mathbb{G}, \mathbb{Z}_q\}_{i\in[\ell]}$ corresponding to witness $\{\mathsf{sk}_i, b_i\}_{i\in[\ell]}$.
6. **Sender generates OT message, Rerandomizes and Permutes Receiver's Public Keys:** The sender aborts if $T \neq \mathsf{Com}_{\mathbb{G}}(Q_{\mathsf{R}}; t)$ else it sets $Q = Q_{\mathsf{S}} \cdot Q_{\mathsf{R}}$. The sender samples random choice bit permutation $\boldsymbol{\rho} \leftarrow_R \{0,1\}^\ell$. For $i \in [\ell]$ the sender performs the following:
   - The sender computes $\mathsf{pk}_{i,1} = \frac{Q}{\mathsf{pk}_{i,0}}$.
   - The sender samples $r_{i,0}, r_{i,1} \leftarrow_R \mathbb{Z}_q$.
   - The sender computes $R_{i,0} = g^{r_{i,0}}$ and $R_{i,1} = g^{r_{i,1}}$.

   The sender sends $(\boldsymbol{\rho}, \{R_{i,0}, R_{i,1}, p_i, d^{\mathsf{S}}_i\}_{i\in[\ell]})$ to the receiver.
7. **Receiver Rerandomizes Sender's OT message and Computes Output:** The receiver sets the random choice bit string as $\mathbf{s} = \mathbf{b} \oplus \boldsymbol{\rho}$. For $i \in [\ell]$, the receiver performs the following:
   - The receiver aborts if $c^{\mathsf{S}}_i \neq \mathsf{Com}_q(p_i; d^{\mathsf{S}}_i)$.
   - The receiver sets $p_{i,0} = p_i$ and $p_{i,1} = -p_i$.
   - The receiver updates $\mathsf{sk}_i = \mathsf{sk}_i + p_{i,b_i}$ and computes $k'_i = (R_{i,s_i} \cdot g^{v_{i,s_i}})^{\mathsf{sk}_i}$.

   The receiver outputs $(\mathbf{s}, \mathbf{k}')$ where $\mathbf{k}' = \{k'_i\}_{i\in[\ell]}$. The receiver decommits $(\mathbf{C}^{\mathsf{R}}_0, \mathbf{C}^{\mathsf{R}}_1)$ by sending $\{v_{i,0}, d^{\mathsf{R}}_{i,0}, v_{i,1}, d^{\mathsf{R}}_{i,1}\}_{i\in[\ell]}$ to sender.

Fig. 10: Protocol π_{rOT} implementing $\mathcal{F}_{\mathsf{rOT}}$

8. **Sender Computes Rerandomized Output:** For $i \in [\ell]$ the sender computes the following:
   - For $\beta \in \{0,1\}$: The sender aborts if $c^{\mathsf{R}}_{i,\beta} \neq \mathsf{Com}_q(v_{i,\beta}; d^{\mathsf{R}}_{i,\beta})$.
   - Sets $p_{i,0} = p_i$ and $p_{i,1} = -p_i$.
   - The sender computes $k_{i,0}$ and $k_{i,1}$ based on ρ_i by considering the following two cases:
     - If $(\rho_i == 0)$: the sender computes the output messages $k_{i,0} = (\mathsf{pk}_{i,0} \cdot g^{p_{i,0}})^{r_{i,0}+v_{i,0}}$ and $k_{i,1} = (\mathsf{pk}_{i,1} \cdot g^{p_{i,1}})^{r_{i,1}+v_{i,1}}$.
     - If $(\rho_i == 1)$: the sender computes the output messages $k_{i,0} = (\mathsf{pk}_{i,1} \cdot g^{p_{i,1}})^{r_{i,0}+v_{i,0}}$ and $k_{i,1} = (\mathsf{pk}_{i,0} \cdot g^{p_{i,0}})^{r_{i,1}+v_{i,1}}$.

     More generally, the sender computes $k_{i,0} = (\mathsf{pk}_{i,\rho_i} \cdot g^{p_{i,\rho_i}})^{r_{i,0}+v_{i,0}}$ and $k_{i,1} = (\mathsf{pk}_{i,\overline{\rho_i}} \cdot g^{p_{i,\overline{\rho_i}}})^{r_{i,1}+v_{i,1}}$.

   The sender sets $(\mathbf{k}_0, \mathbf{k}_1) = \{k_{i,0}, k_{i,1}\}_{i\in[\ell]}$ as the output.

Fig. 11: Protocol π_{rOT} implementing $\mathcal{F}_{\mathsf{rOT}}$

1. **Receiver's Coin-tossing for Parameters:** The simulator samples $Q_{\mathsf{R}} \leftarrow_R \mathbb{G}$ and sends $T = \mathsf{Com}_{\mathbb{G}}(Q_{\mathsf{R}}; t)$ to the sender.
2. **Sender's Coin-tossing for Parameters and Receiver's Public Key:** The sender sends $(Q_{\mathsf{S}}, \mathbf{C}^{\mathsf{S}})$, where $\mathbf{C}^{\mathsf{S}} = \{c_i^{\mathsf{S}}\}_{i\in[\ell]}$, to the simulated receiver.
3. **Sender's Zero-Knowledge Proof for Parameters:** The sender and the simulated receiver run $\pi_{\mathsf{ZK}}^{\mathsf{DL}}$ protocol where sender is the prover for the statement $(Q_{\mathsf{S}}, \mathbb{G}, \mathbb{Z}_q)$. The simulator invokes the simulator for $\pi_{\mathsf{ZK}}^{\mathsf{DL}}$ to extract the witness q. The simulator aborts if it fails to obtain a correct witness.
4. **Receiver's generates Public Keys and Performs Coin-tossing for Sender's OT message:** The simulator computes $Q = Q_{\mathsf{R}} \cdot Q_{\mathsf{S}}$. The simulator samples $\mathbf{b} = \{0,1\}^{\ell}$. For $i \in [\ell]$ the simulator performs the following:
   - The simulator samples $\mathsf{sk}_{i,0} \leftarrow_R \mathbb{Z}_q$ and computes $\mathsf{pk}_{i,0} = g^{\mathsf{sk}_{i,0}}$.
   - The simulator sets $\mathsf{sk}_{i,1} = q - \mathsf{sk}_{i,0}$ and computes $\mathsf{pk}_{i,1} = \frac{Q}{\mathsf{pk}_{i,0}}$.
   - The simulator samples shares for sender's OT randomness $v_{i,0}, v_{i,1} \leftarrow_R \mathbb{Z}_q$.
   - The simulator commits to the shares as $c_{i,0}^{\mathsf{R}} = \mathsf{Com}_q(v_{i,0}; d_{i,0}^{\mathsf{R}})$ and $c_{i,1}^{\mathsf{R}} = \mathsf{Com}_q(v_{i,1}; d_{i,1}^{\mathsf{R}})$.

   The simulator decommits to T by sending (Q_{R}, t). The simulator also sends the commitments - $(\mathbf{C}_0^{\mathsf{R}}, \mathbf{C}_1^{\mathsf{R}})$ where $\mathbf{C}_0^{\mathsf{R}} = \{c_{i,0}^{\mathsf{R}}\}_{i\in[\ell]}$ and $\mathbf{C}_1^{\mathsf{R}} = \{c_{i,1}^{\mathsf{R}}\}_{i\in[\ell]}$ and the public keys $\{\mathsf{pk}_{i,0}\}_{i\in[\ell]}$.
5. **Receiver's WI Proof for Secret Keys:** For $i \in [\ell]$, the simulated receiver and the sender parallely run $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ protocol where receiver is the prover for the statement $\{\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1}, \mathbb{G}, \mathbb{Z}_q\}_{i\in[\ell]}$ corresponding to witness $\{\mathsf{sk}_{i,0}, 0\}_{i\in[\ell]}$.
6. **Sender generates OT message, Rerandomizes and Permutes Receiver's Public Keys:** The corrupt sender sends $(\boldsymbol{\rho}, \{R_{i,0}, R_{i,1}, p_i, d_i^{\mathsf{S}}\}_{i\in[\ell]})$ to the simulated receiver.
7. **Receiver Rerandomizes Sender's OT message and Computes Output:** The simulator sets the random choice bit string as $\mathbf{s} = \mathbf{b} \oplus \boldsymbol{\rho}$. For $i \in [\ell]$, the simulator performs the following:
   - The simulator aborts if $c_i^{\mathsf{S}} \neq \mathsf{Com}_q(p_i; d_i^{\mathsf{S}})$.
   - The simulator sets $p_{i,0} = p_i$ and $p_{i,1} = -p_i$.
   - The simulator updates $\mathsf{sk}_{i,0} = \mathsf{sk}_{i,\rho_i} + p_{i,\rho_i}$ and $\mathsf{sk}_{i,1} = \mathsf{sk}_{i,\overline{\rho_i}} + p_{i,\overline{\rho_i}}$.
   - The simulator computes $k_{i,0} = (R_{i,0} \cdot g^{v_{i,0}})^{\mathsf{sk}_{i,0}}$ and $k_{i,1} = (R_{i,1} \cdot g^{v_{i,1}})^{\mathsf{sk}_{i,1}}$.

   The simulator sets $(\mathbf{k}_0, \mathbf{k}_1) = \{k_{i,0}, k_{i,1}\}_{i\in[\ell]}$ and invokes $\mathcal{F}_{\mathsf{rOT}}$ with input $(\textsc{Initiate}, \mathsf{sid}, \ell)$ and corrupt sender inputs - $(\mathbf{k}_0, \mathbf{k}_1)$. The simulator decommits $(\mathbf{C}_0^{\mathsf{R}}, \mathbf{C}_1^{\mathsf{R}})$ by sending $\{v_{i,0}, d_{i,0}^{\mathsf{R}}, v_{i,1}, d_{i,1}^{\mathsf{R}}\}_{i\in[\ell]}$ to sender.
8. **Sender Computes Rerandomized Output:** The sender performs its own adversarial strategy.

Fig. 12: Simulation against a corrupt sender in π_{rOT}

1. **Receiver's Coin-tossing for Parameters:** The receiver sends $T = \mathsf{Com}_{\mathbb{G}}(Q_{\mathsf{R}}; t)$ to the sender.
2. **Sender's Coin-tossing for Parameters and Receiver's Public Key:** The simulator samples $q \leftarrow_R \mathbb{Z}_q$ and computes $Q_{\mathsf{S}} = g^q$. For $i \in [\ell]$ the simulator performs the following:
   - The simulator samples $p_i \leftarrow_R \mathbb{Z}_q$ to rerandomize the receiver public keys.
   - The simulator computes $c_i^{\mathsf{S}} = \mathsf{Com}_q(p_i; d_i^{\mathsf{S}})$.

   The simulator sends $(Q_{\mathsf{S}}, \mathbf{C}^{\mathsf{S}})$ to the receiver, where $\mathbf{C}^{\mathsf{S}} = \{c_i^{\mathsf{S}}\}_{i\in[\ell]}$.
3. **Sender's Zero-Knowledge Proof for Parameters:** The simulator invokes the ZK simulator of $\pi_{\mathsf{ZK}}^{\mathsf{DL}}$ to simulate the proof for the statement $(Q_{\mathsf{S}}, \mathbb{G}, \mathbb{Z}_q)$.
4. **Receiver's generates Public Keys and Performs Coin-tossing for Sender's OT message:** The receiver decommits to T by sending (Q_{R}, t). The receiver also sends the commitments - $(\mathbf{C}_0^{\mathsf{R}}, \mathbf{C}_1^{\mathsf{R}})$ where $\mathbf{C}_0^{\mathsf{R}} = \{c_{i,0}^{\mathsf{R}}\}_{i\in[\ell]}$ and $\mathbf{C}_1^{\mathsf{R}} = \{c_{i,1}^{\mathsf{R}}\}_{i\in[\ell]}$ and the public keys $\{\mathsf{pk}_{i,0}\}_{i\in[\ell]}$.
5. **Receiver's WI Proof for Secret Keys:** For $i \in [\ell]$, the receiver and the simulated sender parallely run $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ protocol where receiver is the prover for the statement $\{\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1}, \mathbb{G}, \mathbb{Z}_q\}_{i\in[\ell]}$. The simulator either extracts the witness $\{\mathsf{sk}_i, b_i\}_{i\in[\ell]}$ or it aborts if extraction fails for any statement.
6. **Sender generates OT message, Rerandomizes and Permutes Receiver's Public Keys:** The simulator aborts if $T \neq \mathsf{Com}_{\mathbb{G}}(Q_{\mathsf{R}}; t)$ else it sets $Q = Q_{\mathsf{S}} \cdot Q_{\mathsf{R}}$. The simulator samples random choice bit permutation $\boldsymbol{\rho} \leftarrow_R \{0,1\}^{\ell}$. For $i \in [\ell]$ the simulator performs the following:
   - The simulator computes $\mathsf{pk}_{i,1} = \frac{Q}{\mathsf{pk}_{i,0}}$.
   - The simulator samples $r_{i,0}, r_{i,1} \leftarrow_R \mathbb{Z}_q$.
   - The simulator computes $R_{i,0} = g^{r_{i,0}}$ and $R_{i,1} = g^{r_{i,1}}$.

   The simulator sends $(\boldsymbol{\rho}, \{R_{i,0}, R_{i,1}, p_i, d_i^{\mathsf{S}}\}_{i\in[\ell]})$ to the receiver.
7. **Receiver Rerandomizes Sender's OT message and Computes Output:** The receiver decommits $(\mathbf{C}_0^{\mathsf{R}}, \mathbf{C}_1^{\mathsf{R}})$ by sending $\{v_{i,0}, d_{i,0}^{\mathsf{R}}, v_{i,1}, d_{i,1}^{\mathsf{R}}\}_{i\in[\ell]}$ to the simulated sender.
8. **Sender Computes Rerandomized Output:** For $i \in [\ell]$ the simulator computes the following:
   - For $\beta \in \{0,1\}$: The simulator aborts if $c_{i,\beta}^{\mathsf{R}} \neq \mathsf{Com}_q(v_{i,\beta}; d_{i,\beta}^{\mathsf{R}})$.
   - Sets $p_{i,0} = p_i$ and $p_{i,1} = -p_i$.
   - If $\rho_i == 0$: the simulator computes the output messages $k_{i,b_i} = (\mathsf{pk}_{i,b_i} \cdot g^{p_{i,b_i}})^{r_{i,b_i}+v_{i,b_i}}$ and $k_{i,\overline{b_i}} \leftarrow_R \mathbb{G}$.
   - If $\rho_i == 1$: the simulator computes the output messages $k_{i,\overline{b_i}} = (\mathsf{pk}_{i,b_i} \cdot g^{p_{i,b_i}})^{r_{i,\overline{b_i}}+v_{i,\overline{b_i}}}$ and $k_{i,b_i} \leftarrow_R \mathbb{G}$.

   The simulator sets $(\mathbf{k}_0, \mathbf{k}_1) = \{k_{i,0}, k_{i,1}\}_{i\in[\ell]}$ as the sender OT output. The simulator sets $\mathbf{s} = (b_i \oplus \rho_i)_{i\in[\ell]}$ and and $\mathbf{k}' = \{k_i'\}_{i\in[\ell]} = \{k_{i,s_i}\}_{i\in[\ell]}$. The simulator invokes $\mathcal{F}_{\mathsf{rOT}}$ with input $(\textsc{Initiate}, \mathsf{sid}, \ell)$ and corrupt receiver inputs - $(\mathbf{s}, \mathbf{k}')$.

Fig. 13: Simulation against a corrupt receiver in π_{rOT}

– Hyb_1 : Same as Hyb_0, except the simulator invokes π_{ZK} the ZK simulator of $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ to simulate the proof for the statement $(Q_{\mathsf{S}}, \mathbb{G}, \mathbb{Z}_q)$. Indistinguishability follows from simulation based security of $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ against a corrupt verifier. The zero knowledge property of $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ ensures this indistinguishability.

– Hyb_2 : Same as Hyb_1, except the simulator extracts the corrupt receiver's witnesses - $\{\mathsf{sk}_i, b_i\}$ for $i \in [\ell]$ from the WI proofs. Indistinguishability follows from the proof of knowledge property of $\pi^{\mathsf{OR}}_{\mathsf{WI}}$.

– Hyb_3 : Same as Hyb_2, except the simulator samples $k_{i,\overline{b_i}} \leftarrow_R \mathbb{G}$ randomly and constructs $(\mathbf{k}_0, \mathbf{k}_1)$ following the simulation algorithm. It invokes $\mathcal{F}_{\mathsf{rOT}}$ with corrupt receiver's input as $(\mathbf{s}, \mathbf{k}')$. Indistinguishability follows from the binding of T (i.e. $\mathsf{Com}_{\mathbb{G}}$), binding of $(\mathbf{c}^{\mathsf{R}}_0, \mathbf{c}^{\mathsf{R}}_1)$ (i.e. Com_q) and the DDH assumption. A corrupt receiver distinguishing between an honestly constructed $k_{i,\overline{s_i}}$ and a simulated $k_{i,\overline{s_i}} \leftarrow_R \mathbb{G}$ is either able to break the DDH assumption or able to bias the distribution by biasing the distribution of Q or $R_{i,\overline{b_i}} \cdot g^{v_{i,\overline{b_i}}}$. Such an adversary can be used to break binding of the commitment schemes. The reduction for the adversary of $\mathsf{Com}_{\mathbb{G}}$ runs the simulation algorithm against a corrupt receiver until step 4 to receive the opening (Q_{R}, t) corresponding to commitment T. Then it rewinds to Step 2 and reruns the simulation algorithm with different randomness. If it obtains openings valid $(Q'_{\mathsf{R}}, t') \neq (Q_{\mathsf{R}}, t)$ for the same commitment T then it returns the response - $T, (Q_{\mathsf{R}}, t), (Q'_{\mathsf{R}}, t')$, to the challenger (for the binding game of $\mathsf{Com}_{\mathbb{G}}$). We follow the same roadmap for constructing an adversary for the binding property of Com_q. The reduction for the adversary of Com_q runs the simulation algorithm against a corrupt receiver until step 8 to receive the opening $(v_{i,\beta}, d^{\mathsf{R}}_{i,\beta})$ corresponding to commitment $c^{\mathsf{R}}_{i,\beta}$ for $\beta \in \{0,1\}$. Then it rewinds to Step 2 and reruns the simulation algorithm with different randomness. If it obtains openings valid $(v'_{i,\beta}, d^{\mathsf{R}\,\prime}_{i,\beta}) \neq (v'_{i,\beta}, d^{\mathsf{R}\,\prime}_{i,\beta})$ corresponding to the same commitment $c^{\mathsf{R}}_{i,\beta}$ for $\beta \in \{0,1\}$ then it returns the response - $c^{\mathsf{R}}_{i,\beta}, (v'_{i,\beta}, d^{\mathsf{R}\,\prime}_{i,\beta}) \neq (v'_{i,\beta}, d^{\mathsf{R}\,\prime}_{i,\beta})$), to the challenger (for the binding game of Com_q). Finally, we construct an adversary for DDH given a distinguisher for the two hybrids. The DDH adversary receives (g, X, Y, Z) as the DDH challenge. It simulates by rewinding step 4 such that $Q = X$. The reduction extracts b_i and sets $R_{i,\overline{b_i}} = Y$ and sets $k_{i,\overline{s_i}} = \frac{Z}{Y^{\mathsf{sk}_i}} \cdot Y^{v_{i,\overline{b_i}}}$. The DDH adversary returns whatever the distinguisher outputs. If $Z = X^y$ where $y = g^y$ then the receiver is in Hyb_2, else it is in Hyb_3. Hence, a distinguisher for the hybrids can be successfully used to break the DDH assumption.

□

We provide the reverse firewall $\mathsf{RF}_{\mathsf{rOT}}$ for protocol π_{rOT} in in Appendix. **??**. We show that the firewall maintains functionality and provides ER for a tampered sender against a receiver and also provides ER for a tampered receiver against a sender by proving Thm. 7.

Theorem 7. *Assuming $\mathsf{Com}_{\mathbb{G}}$ and Com_q be computationally binding and hiding commitment schemes where they are rerandomizable and additively homomorphic for message spaces over $\mathbb{G}$ and $\mathbb{Z}_q$ elements respectively, $\mathsf{RF}_{\mathsf{ZK}}$ and $\mathsf{RF}_{\mathsf{WI}}$ provides exfiltration resistance for the tampered parties in $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ and $\pi^{\mathsf{OR}}_{\mathsf{WI}}$ respectively, then the above firewall $\mathsf{RF}_{\mathsf{rOT}}$ provides exfiltration resistance for a tampered sender against a receiver, and for a tampered receiver against a sender.*

Proof. The firewall $\mathsf{RF}_{\mathsf{rOT}}$ is presented as follows. It is same for the sender and the receiver and hence we present only for one of them.

1. Upon receiving T, $\mathsf{RF}_{\mathsf{rOT}}$ samples $\widetilde{q} \leftarrow_R \mathbb{Z}_q$ and computes $\widehat{T} = T \cdot \mathsf{Com}_{\mathbb{G}}(g^{\widetilde{q}}; \widetilde{t})$ where $\widetilde{t} \leftarrow_R \{0,1\}^*$. The $\mathsf{RF}_{\mathsf{rOT}}$ sends $\widehat{T}$ to the sender.
2. Upon receiving $(Q_{\mathsf{S}}, \mathbf{C}^{\mathsf{S}})$ from sender, the firewall sets $\widehat{Q_{\mathsf{S}}} = Q_{\mathsf{S}} \cdot g^{\widetilde{q}}$ For $i \in [\ell]$, $\mathsf{RF}_{\mathsf{rOT}}$ performs the following for $i \in [\ell]$:
   – $\mathsf{RF}_{\mathsf{rOT}}$ samples $\widetilde{p_i}$ to sanitize sender's randomness p_i.
   – The firewall also samples a permutation bit $\alpha_i \leftarrow_R \{0,1\}$.
   – The firewall sanitizes c^{S}_i as $\widehat{c^{\mathsf{S}}_i}$ based on α_i as follows:
     • $(\alpha_i == 0)$: Set $\widehat{c^{\mathsf{S}}_i} = c^{\mathsf{S}}_i \cdot \mathsf{Com}_q(\widetilde{p_i}; \widetilde{d^{\mathsf{S}}_i})$ where $\widetilde{d^{\mathsf{S}}_i} \leftarrow_R \{0,1\}^*$.
     • $(\alpha_i == 1)$: Set $\widehat{c^{\mathsf{S}}_i} = \mathsf{Com}_q(-\widetilde{p_i}; \widetilde{d^{\mathsf{S}}_i}) \cdot (c^{\mathsf{S}}_i)^{-1}$ where $\widetilde{d^{\mathsf{S}}_i} \leftarrow_R \{0,1\}^*$.

   $\mathsf{RF}_{\mathsf{rOT}}$ sends $(\widehat{Q_{\mathsf{S}}}, \widehat{\mathbf{C}^{\mathsf{S}}})$ where $\widehat{\mathbf{C}^{\mathsf{S}}} = \{\widehat{c^{\mathsf{S}}_i}\}_{i \in [\ell]}$ to the receiver.
3. $\mathsf{RF}_{\mathsf{rOT}}$ invokes the firewall $\mathsf{RF}_{\mathsf{ZK}}(\widetilde{q})$ (Fig. 17) for the ZK protocol to rerandomize the transcript of $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$.

4. Upon receiving $(Q_{\mathsf{R}}, t, (\mathbf{C}_0^{\mathsf{R}}, \mathbf{C}_1^{\mathsf{R}}), \{\mathsf{pk}_{i,0}\}_{i\in[\ell]}$ from the receiver the firewall sets $\widehat{Q_{\mathsf{R}}} = Q_{\mathsf{R}} \cdot g^{\widetilde{q}}$, $\widehat{t} = t + \widetilde{t}$ and computes $\widehat{Q} = Q_{\mathsf{S}} \cdot Q_{\mathsf{R}} \cdot g^{\widetilde{q}}$. In addition the firewall performs the following for $i \in [\ell]$:
   - $\mathsf{RF}_{\mathsf{rOT}}$ samples $\widetilde{v_{i,0}}, \widetilde{v_{i,1}} \leftarrow_R \mathbb{Z}_q$. $\mathsf{RF}_{\mathsf{rOT}}$ computes $\widehat{c_{i,0}^{\mathsf{R}}} = c_{i,0}^{\mathsf{R}} \cdot \mathsf{Com}_q(\widetilde{v_{i,0}}; \widetilde{d_{i,0}^{\mathsf{R}}})$ and $\widehat{c_{i,1}^{\mathsf{R}}} = c_{i,1}^{\mathsf{R}} \cdot \mathsf{Com}_q(\widetilde{v_{i,1}}; \widetilde{d_{i,1}^{\mathsf{R}}})$ where $\widetilde{d_{i,0}^{\mathsf{R}}}, \widetilde{d_{i,1}^{\mathsf{R}}} \leftarrow_R \{0,1\}^*$.
   - Based on the permutation bit α_i the firewall performs the following:
     - $(\alpha_i == 0)$: Set $\widehat{\mathsf{pk}_{i,0}} = \mathsf{pk}_{i,0} \cdot g^{\widetilde{p_i}}$.
     - $(\alpha_i == 1)$: Set $\widehat{\mathsf{pk}_{i,0}} = \mathsf{pk}_{i,1} \cdot g^{\widetilde{p_i}} = \frac{\widehat{Q}}{\mathsf{pk}_{i,0}} \cdot g^{\widetilde{p_i}}$.

   $\mathsf{RF}_{\mathsf{rOT}}$ sends $(\widehat{Q_{\mathsf{R}}}, \widehat{t}, (\widehat{\mathbf{C}_0^{\mathsf{R}}}, \widehat{\mathbf{C}_1^{\mathsf{R}}}), \{\widehat{\mathsf{pk}_{i,0}}\}_{i\in[\ell]})$ to the sender where $\widehat{\mathbf{C}_0^{\mathsf{R}}} = \{\widehat{c_{i,0}^{\mathsf{R}}}\}_{i\in[\ell]}$ and $\widehat{\mathbf{C}_1^{\mathsf{R}}} = \{\widehat{c_{i,1}^{\mathsf{R}}}\}_{i\in[\ell]}$.
5. For $i \in [\ell]$, $\mathsf{RF}_{\mathsf{rOT}}$ invokes the firewall $\mathsf{RF}_{\mathsf{WI}}((\widetilde{p_i}, -\widetilde{p_i}), \alpha_i)$ (Fig. 18) of the WI protocol for OR composition to rerandomize the transcript of $\pi_{\mathsf{WI}}^{\mathsf{OR}}$.
6. Upon receiving $\{\rho_i, R_{i,0}, R_{i,1}, p_i, d_i^{\mathsf{S}},\}_{i\in[\ell]}$ from the sender, the firewall performs the following for $i \in [\ell]$:
   - Set $\widehat{\rho_i} = \rho_i \oplus \alpha_i$.
   - Set $\widehat{R_{i,0}} = R_{i,0} \cdot g^{\widetilde{v_{i,0}}}$ and $\widehat{R_{i,1}} = R_{i,1} \cdot g^{\widetilde{v_{i,1}}}$.
   - Based on permutation bit α_i the firewall performs the following:
     - $(\alpha_i == 0)$: Set $\widehat{p_i} = p_i + \widetilde{p_i}$ and $\widehat{d_i^{\mathsf{S}}} = d_i^{\mathsf{S}} + \widetilde{d_i^{\mathsf{S}}}$.
     - $(\alpha_i == 1)$: Set $\widehat{p_i} = -p_i - \widetilde{p_i}$ and $\widehat{d_i^{\mathsf{S}}} = \widetilde{d_i^{\mathsf{S}}} - d_i^{\mathsf{S}}$.

   The firewall sends $\{\widehat{\rho_i}, \widehat{R_{i,0}}, \widehat{R_{i,1}}, \widehat{p_i}, \widehat{d_i^{\mathsf{S}}},\}_{i\in[\ell]}$ to the receiver.
7. Upon receiving $\{v_{i,0}, d_{i,0}^{\mathsf{R}}, v_{i,1}, d_{i,1}^{\mathsf{R}}\}_{i\in[\ell]}$ from the receiver, the firewall performs the following for $i \in [\ell]$:
   - Set $\widehat{v_{i,0}} = v_{i,0} + \widetilde{v_{i,0}}$ and $\widehat{v_{i,1}} = v_{i,1} + \widetilde{v_{i,1}}$.
   - Set $\widehat{d_{i,0}^{\mathsf{R}}} = d_{i,0}^{\mathsf{R}} + \widetilde{d_{i,0}^{\mathsf{R}}}$ and $\widehat{d_{i,1}^{\mathsf{R}}} = d_{i,1}^{\mathsf{R}} + \widetilde{d_{i,1}^{\mathsf{R}}}$.

   The firewall sends $\{\widehat{v_{i,0}}, \widehat{d_{i,0}^{\mathsf{R}}}, \widehat{v_{i,1}}, \widehat{d_{i,1}^{\mathsf{R}}}\}_{i\in[\ell]}$ to the sender.

We show that $\mathsf{RF}_{\mathsf{rOT}}$ is correct and maintaining functionality of π_{rOT}. The correctness of the firewall can be verified for $(\alpha_i == 0)$ and $(\alpha_i == 1)$ such that $\widehat{k_i'}$ outputted by the receiver is same as $\widehat{k_{i,\widehat{s_i}}}$.

- $(\alpha_i == 0, \rho_i == 0)$: The sender outputs $\widehat{k_{i,0}} = (\mathsf{pk}_{i,0} \cdot g^{p_i+\widetilde{p_i}})^{r_{i,0}+v_{i,0}+\widetilde{v_{i,0}}}$ and $\widehat{k_{i,1}} = (\mathsf{pk}_{i,1} \cdot g^{-(p_i+\widetilde{p_i})})^{r_{i,1}+v_{i,1}+\widetilde{v_{i,1}}}$. The receiver outputs $\widehat{s_i} = b_i$ and $\widehat{k_i'} = (R_{i,\widehat{s_i}} \cdot g^{v_{i,\widehat{s_i}}+\widetilde{v_{i,\widehat{s_i}}}})^{\mathsf{sk}+(-1)^{b_i}(p_i+\widetilde{p_i})})$.
- $(\alpha_i == 0, \rho_i == 1)$: The sender outputs $\widehat{k_{i,0}} = (\mathsf{pk}_{i,1} \cdot g^{p_i+\widetilde{p_i}})^{r_{i,0}+v_{i,0}+\widetilde{v_{i,0}}}$ and $\widehat{k_{i,1}} = (\mathsf{pk}_{i,0} \cdot g^{-(p_i+\widetilde{p_i})})^{r_{i,1}+v_{i,1}+\widetilde{v_{i,1}}}$. The receiver outputs $\widehat{s_i} = b_i \oplus 1$ and $\widehat{k_i'} = (R_{i,\widehat{s_i}} \cdot g^{v_{i,\widehat{s_i}}+\widetilde{v_{i,\widehat{s_i}}}})^{\mathsf{sk}+(-1)^{b_i+1}(p_i+\widetilde{p_i})})$.
- $(\alpha_i == 1, \rho_i == 0)$: The sender outputs $\widehat{k_{i,0}} = (\mathsf{pk}_{i,1} \cdot g^{p_i+\widetilde{p_i}})^{r_{i,0}+v_{i,0}+\widetilde{v_{i,0}}}$ and $\widehat{k_{i,1}} = (\mathsf{pk}_{i,0} \cdot g^{-(p_i+\widetilde{p_i})})^{r_{i,1}+v_{i,1}+\widetilde{v_{i,1}}}$. The receiver outputs $\widehat{s_i} = b_i \oplus 1$ and $\widehat{k_i'} = (R_{i,\widehat{s_i}} \cdot g^{v_{i,\widehat{s_i}}+\widetilde{v_{i,\widehat{s_i}}}})^{\mathsf{sk}+(-1)^{b_i+1}(p_i+\widetilde{p_i})})$.
- $(\alpha_i == 1, \rho_i == 1)$: The sender outputs $\widehat{k_{i,0}} = (\mathsf{pk}_{i,0} \cdot g^{-(p_i+\widetilde{p_i})})^{r_{i,0}+v_{i,0}+\widetilde{v_{i,0}}}$ and $\widehat{k_{i,1}} = (\mathsf{pk}_{i,1} \cdot g^{p_i+\widetilde{p_i}})^{r_{i,1}+v_{i,1}+\widetilde{v_{i,1}}}$. The receiver outputs $\widehat{s_i} = b_i$ and $\widehat{k_i'} = (R_{i,\widehat{s_i}} \cdot g^{v_{i,\widehat{s_i}}+\widetilde{v_{i,\widehat{s_i}}}})^{\mathsf{sk}+(-1)^{b_i+1}(p_i+\widetilde{p_i})})$.

We also demonstrate that $\mathsf{RF}_{\mathsf{rOT}}$ provides ER against tampering of the parties in π_{rOT} by proving Thm. 7.

Proof. We argue ER for a party for each step of the protocol as follows. The same argument holds for both parties.

1. $\widehat{T}$ is identically distributed to an honestly sampled T due to rerandomization property of $\mathsf{Com}_{\mathbb{G}}$.
2. $\widehat{Q_{\mathsf{S}}}$ is identically distributed to an honestly sampled Q_{S} due to $g^{\widetilde{q}}$. The sanitised commitments $\widehat{\mathbf{C}^{\mathsf{S}}}$ are identically distributed to an honestly sampled commitments $\mathbf{C}^{\mathsf{S}}$ due to additive homomorphism of Com_q.
3. Exfiltration resistance is achieved corresponding to the ZK protocol transcript of $\pi_{\mathsf{ZK}}^{\mathsf{DL}}$ due to $\mathsf{RF}_{\mathsf{ZK}}$.
4. $(\widehat{\mathbf{C}_0^{\mathsf{R}}}, \widehat{\mathbf{C}_1^{\mathsf{R}}})$ is identically distributed to honestly sampled $(\mathbf{C}_0^{\mathsf{R}}, \mathbf{C}_1^{\mathsf{R}})$ due to additive homomorphism of Com_q. The distribution of $(\widehat{Q_{\mathsf{R}}}, \widehat{t}, \{\widehat{\mathsf{pk}_{i,0}}\}_{i\in[\ell]})$ is identical to honestly sampled $(Q_{\mathsf{R}}, t, \{\mathsf{pk}_{i,0}\}_{i\in[\ell]})$ due to $g^{\widetilde{q}}$, $\widetilde{t}$ and $\{\alpha_i, \widetilde{p_i}\}_{i\in[\ell]}$.

5. Exfiltration resistance is achieved corresponding to the WI protocol transcript of $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ due to $\mathsf{RF}_{\mathsf{WI}}$.
6. $\{\widehat{\rho_i}, \widehat{R_{i,0}}, \widehat{R_{i,1}}, \widehat{p_i}, \widehat{d_i^{\mathsf{S}}},\}_{i\in[\ell]}$ is identically distributed to an honestly generated $\{\rho_i, R_{i,0}, R_{i,1}, p_i, d_i^{\mathsf{S}},\}_{i\in[\ell]}$ due to $\{\alpha_i, g^{\widetilde{v_{i,0}}}, g^{\widetilde{v_{i,1}}}, \widetilde{p_i}\}_{i\in[\ell]}$ and additive homomorphism of Com_q.
7. Upon receiving $\{v_{i,0}, d_{i,0}^{\mathsf{R}}, v_{i,1}, d_{i,1}^{\mathsf{R}}\}_{i\in[\ell]}$ from the receiver, the firewall performs the following for $i \in [\ell]$:
   - Set $\widehat{v_{i,0}} = v_{i,0} + \widetilde{v_{i,0}}$ and $\widehat{v_{i,1}} = v_{i,1} + \widetilde{v_{i,1}}$.
   - Set $\widehat{d_{i,0}^{\mathsf{R}}} = d_{i,0}^{\mathsf{R}} + \widetilde{d_{i,0}^{\mathsf{R}}}$ and $\widehat{d_{i,1}^{\mathsf{R}}} = d_{i,1}^{\mathsf{R}} + \widetilde{d_{i,1}^{\mathsf{R}}}$.
8. $\{\widehat{v_{i,0}}, \widehat{d_{i,0}^{\mathsf{R}}}, \widehat{v_{i,1}}, \widehat{d_{i,1}^{\mathsf{R}}}\}_{i\in[\ell]}$ is identically distributed to $\{v_{i,0}, d_{i,0}^{\mathsf{R}}, v_{i,1}, d_{i,1}^{\mathsf{R}}\}_{i\in[\ell]}$ due to $\{\widetilde{v_{i,0}}, \widetilde{v_{i,1}}\}_{i\in[\ell]}$ and additive homomorphism of Com_q.

□
□

Cost. The protocol π_{rOT} implements $\mathcal{F}_{\mathsf{rOT}}$ by producing ℓ random OT instances. Each random OT instance communicates 13 group elements + 15 field elements + 1 bit, and performs 35 exponentiations.

6 Fully Malleable Sigma Protocols

We denote a Sigma protocol by $\Sigma = (\mathsf{P}, \mathsf{V})$, where P_1 and P_2 are algorithms that compute, respectively, the prover's first message a, and the prover's last message (response) z. Moreover we require the Sigma protocol to be "unique response", i.e., it is infeasible to find two distinct valid responses for a given first message and fixed challenge. Let $\mathcal{A}$ be the space of all possible prover's first messages; membership in $\mathcal{A}$ can be tested efficiently, so the V always outputs $\perp$ when $a \notin \mathcal{A}$. Also, let $\mathcal{C}$ denote the challenge space of the verifier.

6.1 Malleability

The work of [GMV20] defines the notion of malleability. A Sigma protocol is malleable if the prover's first message a can be randomized into $\widehat{a}$ that is distributed identically to the first message of an honest prover. In addition, for any challenge c, given the coins used to randomize a and any response z yielding an accepting transcript $\tau = (a, c, z)$, a balanced response $\widehat{z}$ can be computed such that $(\widehat{a}, c, \widehat{z})$ is also an accepting transcript. In our constructions, we need a stronger notion of malleability: we will need to randomize the *instance* in addition to the transcript.

The sigma protocol for discrete log is presented in Fig. 14. We demonstrate its malleability in Fig. 15. We present the ZK protocol for discrete log in Fig. 16.

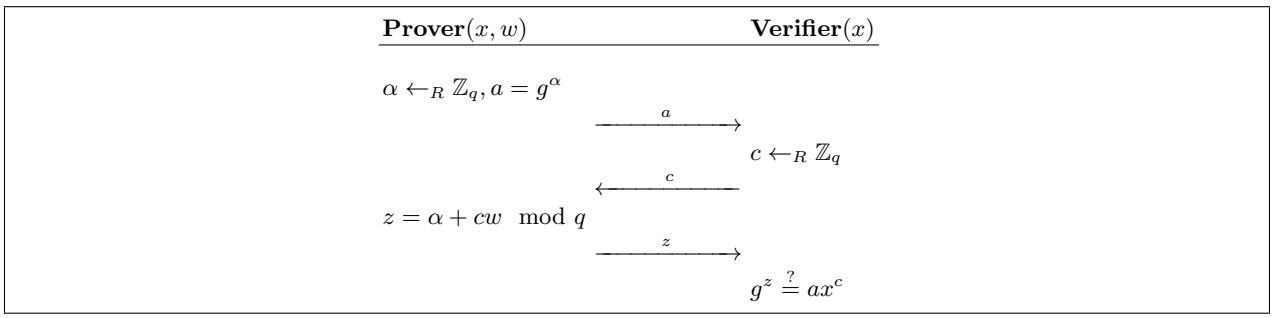

Fig. 14: Sigma protocol for proving knowledge of w such that $x = g^w$

We now formally define our notion of fully malleable Sigma protocols.

Definition 9 (Fully Malleable Sigma protocol). *Let $\Sigma = (\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ be a Sigma protocol for a relation $\mathcal{R}$. Σ is said to be* fully malleable *if there exists a tuple of polynomial-time algorithms* $(\mathsf{Maul}, \mathsf{MaulCh}, \mathsf{Bal})$ *specified as follows:*

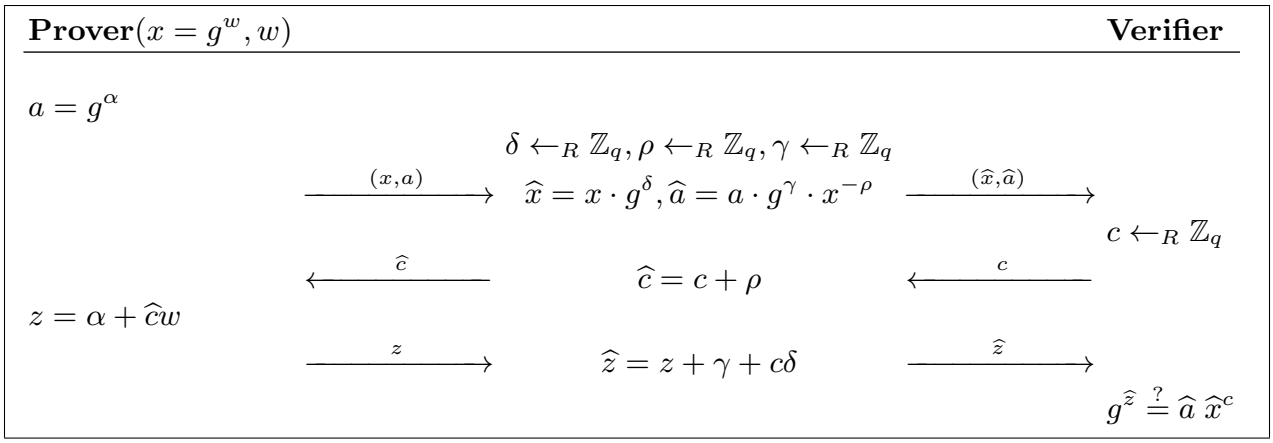

Fig. 15: Fully Malleable Sigma protocol for discrete log

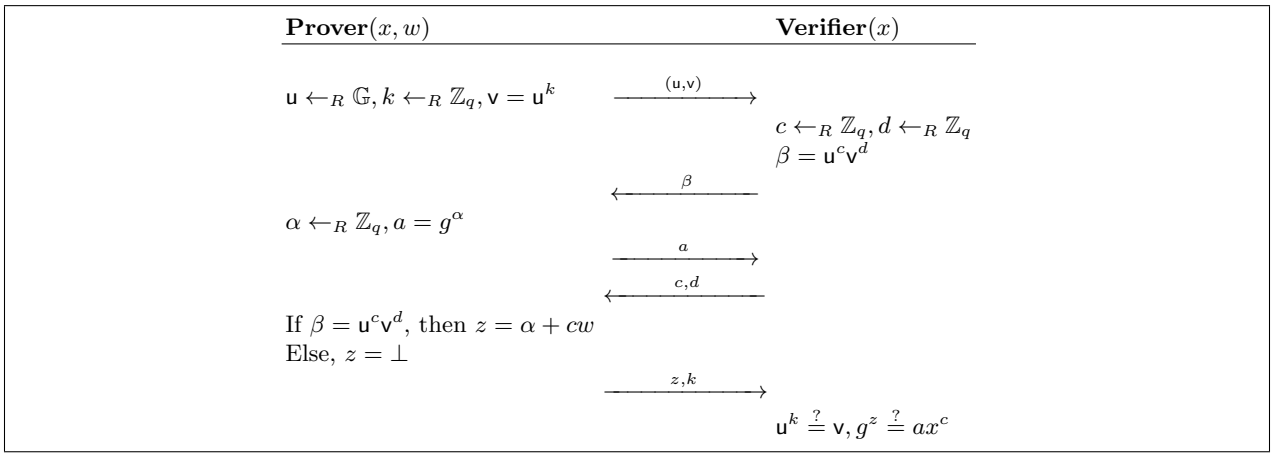

Fig. 16: Sigma protocol $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ compiled to obtain full zero knowledge

(i) Maul *is a probabilistic algorithm that takes as input an instance* x*,* $a \in \mathcal{A}$ *(recall that* $\mathcal{A}$ *is set of all possible prover's first messages), instance randomizer* δ *and outputs an instance* $\widehat{x}$*, and* $\widehat{a} \in \mathcal{A}$ *and state* $\sigma \in \{0,1\}^*$*;*

(ii) MaulCh *is a probabilistic algorithm that takes as input a challenge* c *and a randomizer* ρ *and returns a modified challenge* $\widehat{c}$*.*

(iii) Bal *is a deterministic algorithm that takes as input* x, z*, the state* σ *output by* Maul*, a challenge* $\widehat{c}$ *output by* MaulCh *and returns a balanced response* $\widehat{z}$*.*

The following properties need to be satisfied.

- **Uniformity.** *For all* $(x, w) \in \mathcal{R}$*, and for all* $a \in \mathcal{A}$*,* $\widehat{x}$ *is a uniformly distributed instance in* $\mathcal{L}$*, and the distribution of* $\widehat{a}$ *is identical to that of* $P_1(\widehat{x}, \widehat{w})$*, where* $(\widehat{x}, \widehat{a}, \sigma) \leftarrow_R \mathsf{Maul}(x, a, \delta)$ *such that* $(\widehat{x}, \widehat{w}) \in \mathcal{R}$*. Moreover, for all* $c \in \mathcal{C}$ *(recall that* $\mathcal{C}$ *denotes the challenge space) and uniformly random* $\rho \leftarrow_R \mathbb{Z}_q$*,* $\widehat{c}$ *is uniformly distributed in* $\mathcal{C}$*, where* $\widehat{c} \leftarrow_R \mathsf{MaulCh}(c; \rho)$
- **Malleability.** *For all* $x \in \mathcal{L}$*, for all* $\rho \leftarrow_R \mathbb{Z}_q$ *and for all* $\tau = (a, \widehat{c}, z)$ *such that* $V(x, (a, \widehat{c}, z)) = 1$*, where* $\widehat{c} \leftarrow_R \mathsf{MaulCh}(c; \rho)$*, the following holds :*

$$\Pr[V(\widehat{x}, (\widehat{a}, c, \widehat{z})) = 1 : \ (\widehat{x}, \widehat{a}, \sigma) \leftarrow \mathsf{Maul}(x, a, \delta); \widehat{z} = \mathsf{Bal}(z, \sigma, c)] = 1,$$

where the probability is over the randomness of Maul *and* MaulCh*.*

Lemma 1. *The Sigma protocol in Fig 14 is fully malleable as per Definition 9. The construction is shown in Fig 15.*

Proof. We instantiate Maul, MaulCh and Bal algorithms for knowledge of discrete logarithm, where $\gamma \leftarrow_R \mathbb{Z}_q$:

$$\mathsf{Maul}(x, a, \rho, \delta) = (x \cdot g^{\delta}, a \cdot g^{\gamma} \cdot x^{-\rho}, (\gamma, \delta)) \quad \mathsf{MaulCh}(c, \rho) = c + \rho$$
$$\mathsf{Bal}(z, (\gamma, \delta), c) = z + \gamma + c\delta$$

- *Uniformity:* For all $(x, w), x = g^w$, for all $\alpha \in \mathbb{Z}_q$, the distribution of $\widehat{a} = a \cdot g^{\gamma} \cdot x^{-\rho} = g^{\alpha} \cdot g^{\gamma} \cdot g^{-\rho w}$ over the choice of $\gamma \leftarrow_R \mathbb{Z}_q$ is identical to the distribution of $a = g^{\alpha}$ over the choice of $\alpha \in \mathbb{Z}_q$. Moreover, for all uniformly random $\rho \leftarrow_R \mathbb{Z}_q$, the value $\widehat{c} = c + \rho$ is uniformly distributed in the challenge space.
- *Malleability:* For all $x \in \mathcal{L}$, for all $\rho \in \mathbb{Z}_q$, and for all $\tau = (a, \widehat{c}, z)$ such that $g^z = ax^{-\widehat{c}}$, where $\widehat{c} = c + \rho$, the following holds:

$$\widehat{a}\widehat{x}^{-c} = ag^{\gamma}x^{-\rho}\widehat{x}^{-c} = ag^{\gamma}x^{-c-\rho}g^{-\delta c} = ag^{\gamma}x^{-\widehat{c}}g^{-\delta c} = g^z g^{\gamma} g^{-\delta c} = g^{\widehat{z}}$$

□

We note that Maul and Bal easily generalize to the unifying Sigma protocol for proving knowledge of preimage of a homomorphism [Mau09]. This generalization gives an RF for the unifying Sigma protocol, even though we only need the protocol for knowledge of discrete logarithm in our applications.

In general, Sigma protocols are not full-fledged zero knowledge or zero-knowledge proof of knowledge (ZKPoK) protocols. However, standard techniques [GK96] allow to compile a Sigma protocol into a zero knowledge protocol. We recall the ZKPoK protocol $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ for the discrete logarithm problem in Fig. 16.

The RF for the ZK protocol is shown in Fig. 17 and we show that it provides exfiltration resistant for the parties in $\pi^{\mathsf{DL}}_{\mathsf{ZK}}$ by proving Thm. 8.

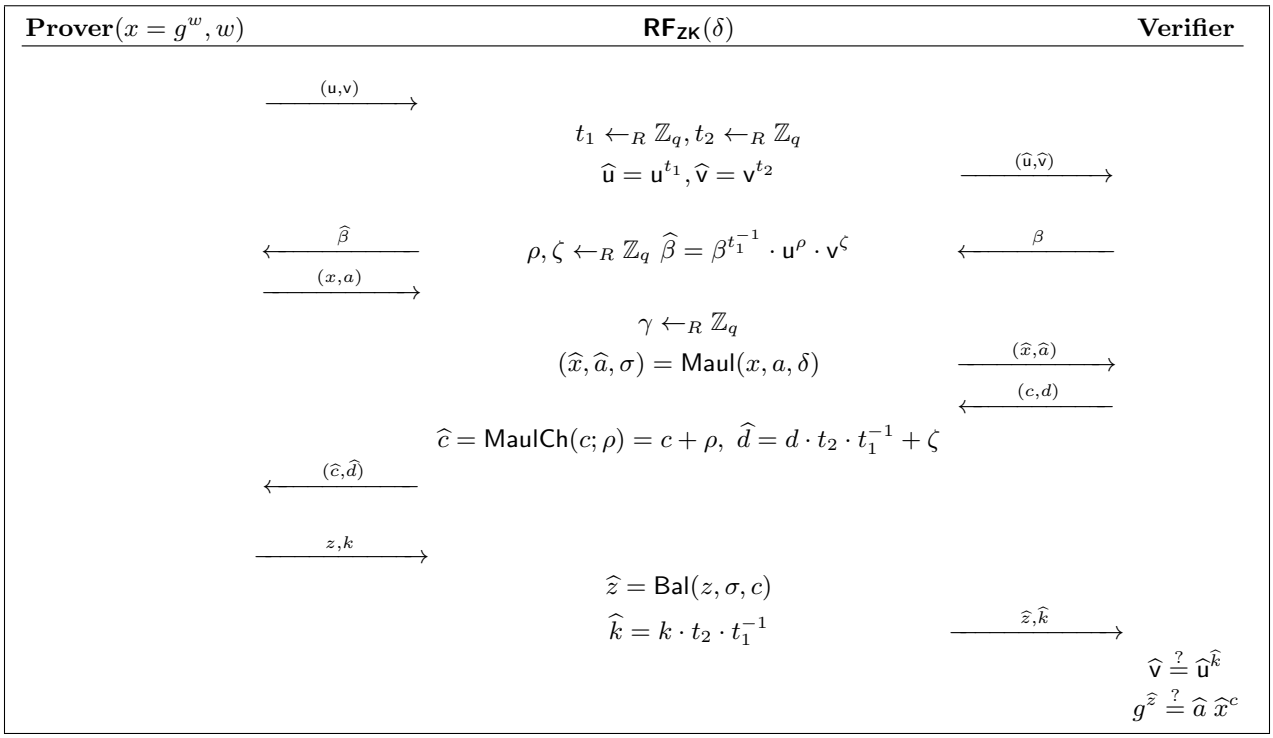

Fig. 17: Reverse Firewall $\mathsf{RF}_{\mathsf{ZK}}$ for ZK compiled Sigma protocol

Theorem 8. *Let Σ be a* fully malleable *unique-response Sigma protocol for $\mathcal{R}$ as in Def 9. The RF $\mathsf{RF}_{\mathsf{ZK}}$ in Fig. 17 is functionality-maintaining, weakly ZK preserving and exfiltration resistant for protocol π^{DL}_{ZK} in Fig 16.*

Proof. Firstly, we note that the protocol in Fig 16 achieves zero-knowledge and proof of knowledge properties [GK96] for discrete logarithm. At a high level, the value β is a Pedersen commitment to the challenge c of the verifier (using randomness d). Since the commitment β is perfectly hiding, the prover gets no information about c and d. The zero-knowledge proof of knowledge (ZKPoK) then follows from the special-soundness property of the underlying Sigma protocol. In particular, the extractor obtains the trapdoor k of the commitment scheme at the end of the protocol and hence can rewind the verifier to the point where the challenge was sent. It then equivocates the commitment β to another challenge c'. At this point, the extractor gets two accepting transcripts (a, c, z) and (a, c', z') and hence can extract the witness w (the discrete logarithm corresponding to x).

It is straight-forward to see that the RF $\mathsf{RF}_{\mathsf{ZK}}$ is functionality-maintaining. In particular, it follows from the malleability of the Perdersen commitment and the full malleability property of the underlying Sigma protocol. We now argue that $\mathsf{RF}_{\mathsf{ZK}}$ provides exfiltration-resistance. By the result of [CGPS21], this also implies (weak) security preservation, in particular weak ZK preservation. The entire transcript of our protocol (Fig 17) is $\tau = (\widehat{\mathsf{u}}, \widehat{\mathsf{v}}, \widehat{\beta}, \widehat{x}, \widehat{a}, \widehat{c}, \widehat{d}, \widehat{z}, \widehat{k})$. The prover receives the messages $(\widehat{\beta}, \widehat{c}, \widehat{d})$ and the verifier receives the messages $(\widehat{\mathsf{u}}, \widehat{\mathsf{v}}, \widehat{x}, \widehat{a}, \widehat{z}, \widehat{k})$. To prove ER, we prove the following:

Lemma 2. *The reverse firewall $\mathsf{RF}_{\mathsf{ZK}}$ for the protocol shown in* Fig 17 *achieves* exfiltration-resistance *for the verifier against the prover.*

Proof. To prove ER for the verifier, we will need to show that the output distribution of an honest verifier V (who follows the protocol honestly) is indistinguishable from the output of the tampered verifier (tampered in a functionality-maintaining way) $\widetilde{\mathsf{V}}$ composed with the RF $\mathsf{RF}_{\mathsf{ZK}}$, i.e., $\widetilde{\mathsf{V}} \circ \mathsf{RF}_{\mathsf{ZK}}$.

The messages output by the verifier in the above protocol is the tuple (β, c, d). After post processing by the RF $\mathsf{RF}_{\mathsf{ZK}}$ the tuple is transformed into the tuple $(\widehat{\beta}, \widehat{c}, \widehat{d})$, which is received the prover. The value $\widehat{\beta}$ is uniformly random even if the commitment β is not uniform. This is because $\widehat{\beta} = \beta^{t_1^{-1}} \cdot \mathsf{u}^{\rho} \cdot \mathsf{v}^{\zeta} = \mathsf{u}^{c+\rho} \mathsf{v}^{t_2 \cdot t_1^{-1} + \zeta}$. Since, the values t_1, t_2, ρ, ζ are sampled uniformly at random by $\mathsf{RF}_{\mathsf{ZK}}$ from $\mathbb{Z}_q$, $\widehat{\beta}$ is also a uniformly random group element, irrespective of β was chosen by $\widetilde{\mathsf{V}}$. Hence, the value $\widehat{\beta}$ is identically distributed to the value β sampled by the honest verifier V. The value $\widehat{c}$ is also a uniformly random element in $\mathbb{Z}_q$. This follows from the *uniformity* property of the algorithm MaulCh of the underlying Σ protocol. Finally, the value $\widehat{d} = d \cdot t_2 \cdot t_1^{-1} + \zeta$ is also a uniformly random element in $\mathbb{Z}_q$, irrespective of how d was chosen by the tampered verifier $\widetilde{\mathsf{V}}$. Hence, the messages $(\widehat{\beta}, \widehat{c}, \widehat{d})$ are distributed identically to the output (β, c, d) of the honest verifier V. □

Lemma 3. *The reverse firewall $\mathsf{RF}_{\mathsf{ZK}}$ for the protocol shown in* Fig 17 *achieves* exfiltration-resistance *for the prover against the verifier.*

Proof. To prove ER for the prover, we will need to show that the output distribution of an honest prover V (who follows the protocol honestly) is indistinguishable from the output of the tampered prover (tampered in a functionality-maintaining way) $\widetilde{\mathsf{P}}$ composed with the RF $\mathsf{RF}_{\mathsf{ZK}}$, i.e., $\widetilde{\mathsf{P}} \circ \mathsf{RF}_{\mathsf{ZK}}$.

The messages output by the prover in the above protocol is $(\mathsf{u}, \mathsf{v}, x, a, z, k)$. After post processing by the RF $\mathsf{RF}_{\mathsf{ZK}}$ the tuple is transformed into the tuple $(\widehat{\mathsf{u}}, \widehat{\mathsf{v}}, \widehat{x}, \widehat{a}, \widehat{z}, \widehat{k})$, which si received by the verifier. Recall that, $\widehat{\mathsf{u}} = \mathsf{u}^{t_1}$ and $\widehat{\mathsf{v}} = \mathsf{v}^{t_2}$, where t_1 and t_2 are sampled independently and uniformly at random from $\mathbb{Z}_q$. Hence, the values $\widehat{\mathsf{u}}$ and $\widehat{\mathsf{v}}$ are uniformly random group elements, irrespective of how u and v were sampled by the tampered prover $\widetilde{\mathsf{P}}$. This implies that the values $\widehat{\mathsf{u}}, \widehat{\mathsf{v}}$ are distributed identically as the values u, and v sampled by the honest prover P. Further, the *uniformity* property of the algorithm Maul (corresponding to the underlying Σ protocol) ensures that the instance $\widehat{x}$ is a uniformly distributed instance in the language and $\widehat{a}$ is identically distributed to the first message computed by an honest prover P (on input $(\widehat{x}, \widehat{w})$ such that $(\widehat{x}, \widehat{w}) \in \mathcal{R}$). Further, since our underlying Σ protocol is a *unique response* protocol, the value of $\widehat{z}$ is determined by the message $\widehat{a}$. Since, the transcript $(\widehat{x}, \widehat{a}, \widehat{z})$ is accepting, the value $\widehat{z}$ is distributed uniformly

subject to the verification condition. Hence $\widehat{z}$ is identically distributed to z sent by an honest prover P. Finally, it is easy to see that the value $\widehat{k}$ is uniformly distributed, irrespective of how k was sampled by the tampered prover $\widehat{\mathsf{P}}$. This is because $\widehat{k} = k \cdot t_2 \cdot t_1^{-1}$, and the values t_2 and t_1^{-1} were sampled independently and uniformly at random from $\mathbb{Z}_q$. Hence, we can conclude that the tuple $(\widehat{\mathsf{u}}, \widehat{\mathsf{v}}, \widehat{x}, \widehat{a}, \widehat{z}, \widehat{k})$ is identically distributed to the tuple $(\mathsf{u}, \mathsf{v}, x, a, z, k)$ sent by an honest prover P in the protocol. □

The proof of Theorem 8 now follows from the proofs of Lemma 3 and Lemma 2 respectively. □

6.2 RF for OR Transform Sigma Protocol

OR Transform. Given x_0, x_1, a prover wishes to prove to a verifier that either $x_0 \in \mathcal{L}_0$ or $x_1 \in \mathcal{L}_1$ without revealing which one is true. The OR relation is given by: $\mathcal{R}_{\mathsf{OR}} = \{((x_0, x_1), w) : (x_0, w) \in \mathcal{R}_0 \vee (x_1, w) \in \mathcal{R}_1\}$.

Let $\Sigma_0 = ((\mathsf{P}_1^0, \mathsf{P}_2^0), \mathsf{V}^0)$ (resp. $\Sigma_1 = ((\mathsf{P}_1^1, \mathsf{P}_2^1), \mathsf{V}^1)$) be a Sigma protocol for language $\mathcal{L}_0$ (resp. $\mathcal{L}_1$). Let Sim^0 (resp. Sim^1) be the HVZK simulator for Σ_0 (resp. Σ_1). A Sigma protocol $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ for the relation $\mathcal{R}_{\mathsf{OR}}$ was constructed in [CDS94]. We describe the protocol $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ in Fig. 19. $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ satisfies perfect special HVZK and perfect WI.

RF for OR Protocol. In order to construct an RF for the OR transform, we need to maul the prover's first message in such a way that the verifier's challenge can be balanced in addition to the prover's last message. We note that [GMV20] considers an RF for the OR composition, however that definition and construction does not suffice for our application since we need to randomize the instance as well. We show the RF for the OR composition in Fig. 18 and demonstrate that it provides ER by proving Thm. 9.

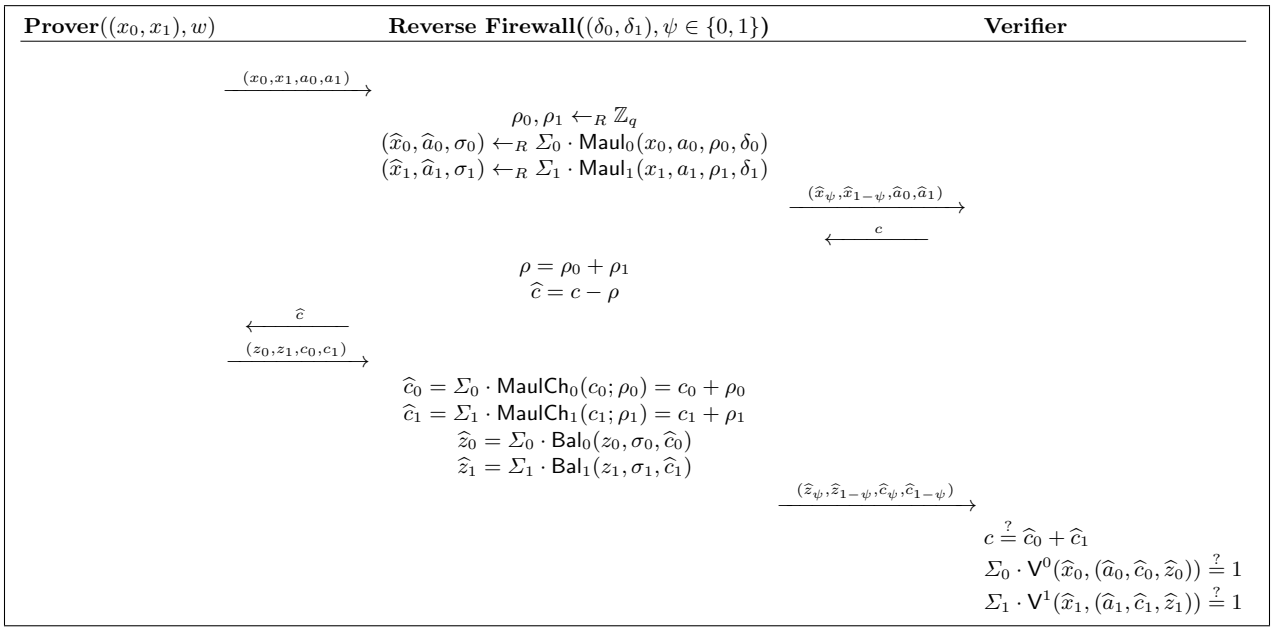

Fig. 18: $\mathsf{RF}_{\mathsf{WI}}$: RF for the OR composition of Sigma protocols, where $(x_b, w) \in \mathcal{R}_b$ for $b \in \{0,1\}$. The bit ψ is an additional input to $\mathsf{RF}_{\mathsf{WI}}$ provided by a RF of an higher-level protocol (in our case the RF of our base OT protocol)

We present the WI protocol for OR composition in Fig. 19 and the corresponding firewall can be found in Fig. 18. We show that the firewall provides ER by proving Thm. 9.

Theorem 9. *Let Σ_0 and Σ_1 be* fully malleable *unique-response Sigma protocols for $\mathcal{R}_0$ and $\mathcal{R}_1$ respectively. The RF $\mathsf{RF}_{\mathsf{WI}}$ in Fig. 18 preserves completeness, is weakly HVZK/WI preserving and exfiltration resistant for π_{WI}^{OR}.*

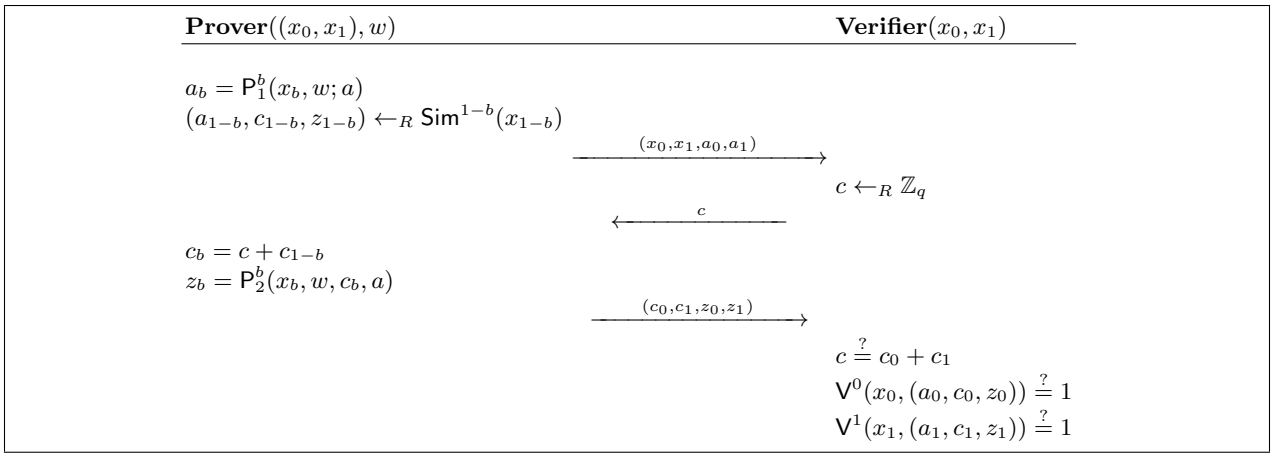

Fig. 19: Protocol $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ for OR composition of Sigma protocols, $b \in \{0,1\}$ is s.t. $(x_b, w) \in \mathcal{R}_b$.

Proof. To proof this theorem, it suffices to prove that the RF $\mathsf{RF}_{\mathsf{WI}}$ is exfiltration-resistant for both the parties. Then, using the result of [CGPS21], this also implies (weak) security preservation, in particular (weak) HVZK/WI preservation. To prove ER we prove the following:

Lemma 4. *The reverse firewall* $\mathsf{RF}_{\mathsf{WI}}$ *for the protocol shown in* Fig 18 *achieves* exfiltration-resistance *for the verifier against the prover.*

Proof. The protocol of the protocol is $\tau = (\widehat{x}_\psi, \widehat{x}_{1-\psi}, \widehat{a}_0, \widehat{a}_1, \widehat{c}, \widehat{z}_\psi, \widehat{z}_{1-\psi}, \widehat{c}_\psi, \widehat{c}_{1-\psi})$. Let us the denote the verifier and the prover of the above protocol Σ_{OR} as $\mathsf{V}_{\mathsf{OR}} = (\mathsf{V}^0, \mathsf{V}^1)$ and $\mathsf{P}_{\mathsf{OR}} = \big(\mathsf{P}^0 = ((\mathsf{P}_1^0, \mathsf{P}_2^0)), \mathsf{P}^1 = (\mathsf{P}_1^1, \mathsf{P}_2^1)\big)$ respectively. To prove ER for the verifier V_{OR}, we will need to show that the output distribution of the honest verifier V_{OR} (who follows the protocol honestly) is indistinguishable from the output of the tampered verifier $\widetilde{\mathsf{V}}_{\mathsf{OR}} = (\widetilde{\mathsf{V}}^0, \widetilde{\mathsf{V}}^1)$ (tampered in a functionality-maintaining way) composed with the RF $\mathsf{RF}_{\mathsf{WI}}$, i.e., $\widetilde{\mathsf{V}}_{\mathsf{OR}} \circ \mathsf{RF}_{\mathsf{WI}}$.

The messages output by the verifier in the above protocol is the only message c. After post processing by the RF $\mathsf{RF}_{\mathsf{ZK}}$ the tuple is transformed into the tuple $\widehat{c} = c - \rho$, which is received the prover P_{OR}. Here $\rho = (\rho_0 + \rho_1)$ is a uniform random element in $\mathbb{Z}_q$ since both ρ_0 and ρ_1 are sampled uniformly at random from $\mathbb{Z}_q$ by $\mathsf{RF}_{\mathsf{WI}}$. Hence, the challenge $\widehat{c}$ is distributed identically to the challenge c sampled by an honest verifier $\widetilde{\mathsf{V}}_{\mathsf{OR}}$. □

Lemma 5. *The reverse firewall* $\mathsf{RF}_{\mathsf{WI}}$ *for the protocol shown in* Fig 18 *achieves* exfiltration-resistance *for the prover against the verifier.*

Proof. To prove ER for the prover P_{OR}, we will need to show that the output distribution of the honest prover P_{OR} (who follows the protocol honestly) is indistinguishable from the output of the tampered prover $\widetilde{\mathsf{P}}_{\mathsf{OR}} = (\widetilde{\mathsf{P}}^0, \widetilde{\mathsf{P}}^1)$ (tampered in a functionality-maintaining way) composed with the RF $\mathsf{RF}_{\mathsf{WI}}$, i.e., $\widetilde{\mathsf{P}}_{\mathsf{OR}} \circ \mathsf{RF}_{\mathsf{WI}}$.

The messages output by the prover in the above protocol is the tuple $(x_0, x_1,$
$a_0, a_1, z_0, z_1, c_0, c_1)$. After post processing by the RF $\mathsf{RF}_{\mathsf{ZK}}$ the tuple is transformed into the tuple $(\widehat{x}_\psi, \widehat{x}_{1-\psi},$ $\widehat{a}_0, \widehat{a}_1, \widehat{c}, \widehat{z}_\psi, \widehat{z}_{1-\psi}, \widehat{c}_\psi, \widehat{c}_{1-\psi})$, which is received by the verifier $\widetilde{\mathsf{V}}_{\mathsf{OR}}$. The uniformity property of the algorithms $\Sigma_0 \cdot \mathsf{Maul}_0$ and $\Sigma_1 \cdot \mathsf{Maul}_1$ (corresponding to the underlying Sigma protocols Σ_0 and Σ_1 respectively) ensures that $\widehat{x}_0$ are $\widehat{x}_1$ are uniformly distributed in the instance space respectively. Additionally, the uniformity property also ensures that the commitments $\widehat{a}_0$ and $\widehat{a}_1$ are identically distributed to the first messages computed by the provers P^0 and P^1. Moreover, since Σ_0 and Σ_1 are unique response Sigma protocols the responses $\widehat{z}_0$ and $\widehat{z}_1$ are determined by $\widehat{a}_0$ and $\widehat{a}_1$ respectively. Since the transcripts $(\widehat{x}_0, \widehat{a}_0, \widehat{z}_0)$ and $(\widehat{x}_1, \widehat{a}_1, \widehat{z}_1)$ are accepting, the values $\widehat{z}_0$ and $\widehat{z}_1$ are distributed uniformly at random subject to the verification conditions. Finally, it is easy to see the challenges $\widehat{c}_0 (= c_0 + \rho_0)$ and $\widehat{c}_1 = (= c_1 + \rho_1)$ are uniformly random challenges

since the values ρ_0 and ρ_1 are sampled uniformly at random from $\mathbb{Z}_q$ by $\mathsf{RF}_{\mathsf{WI}}$. The bit $\psi \in \{0,1\}$ is a uniformly random bit received by $\mathsf{RF}_{\mathsf{WI}}$ and it permutes the instances, responses and challenges according to ψ. This bit ψ is given as input by the RF of our base OT protocol to $\mathsf{RF}_{\mathsf{WI}}$. This bit is used to permute the choice bit of the (tampered) receiver in the OT protocol to ensure that the choice bit is random. □
This completes our proof of Thm. 9. □

7 Quicksilver with Reverse Firewall

We present a variant of Quicksilver [YSWW21] in the firewall setting, π_{QS}, in Fig. 20. It is in the $\mathcal{F}_{\mathsf{cOT}}$ model and provides efficient interactive ZK for binary circuits. For a circuit with number of input wires n and the number of multiplication gate t, the proof size is $(n+t)$ bits in the $\mathcal{F}_{\mathsf{cOT}}$ model. Instantiating $\mathcal{F}_{\mathsf{cOT}}$ with π_{cOT} the concrete proof size of π_{QS} is $(n+t)\kappa + \mathcal{O}(\kappa^2)$ bits. The number of public key operations is $\mathcal{O}(\kappa)$ and is independent of t. Detailed overview can be found in Fig. 2.5. Security of π_{QS} is summarized in Thm. 10.

8 Quicksilver π_{QS} in the Reverse Firewall Setting

We present the quicksilver protocol π_{QS} in the reverse firewall setting in Fig. 20.

Theorem 10. *Assuming H is a collision resistant hash function and Com is a computationally hiding and binding commitment scheme then π_{QS} implements $\mathcal{F}_{ZK}$ functionality in the $\mathcal{F}_{cOT}$ model.*

Proof Sketch. A corrupt prover breaks soundness of the protocol if it 1) breaks binding of $c_{\mathbf{d}}$, or 2) finds a collision in H, or 3) breaks hiding of c_{seed}, or it passes the batch verification phase for a circuit C such that $\forall \mathbf{w}, C(\mathbf{w}) = 0$. Breaking binding of $c_{\mathbf{d}}$ or finding a collision in H allows the prover to open the commitment to a different $\mathbf{d}'$ after obtaining the challenge χ and hence passing the batch verification. Breaking hiding of c_{seed} allows the prover to fix the challenge to a particular value for which it passes the challenge. Finally, assuming the above attacks fail the prover can still pass the batch verification checks if it correctly guesses the entire Δ^κ of the V. The functionality $\mathcal{F}_{\mathsf{cOT}}$ allows the prover to leak c bits of 2^{-c} bits. However, it successfully guesses the entire $\Delta \in \{0,1\}^\kappa$ with $2^{-\kappa}$ probability. Zero knowledge of the protocol follows from the security for a receiver in π_{cOT}. The pads (A_0^*, A_1^*) perfectly hides the inputs of the prover and the ZK simulator simulates the proof given corrupt verifier's input Δ to $\mathcal{F}_{\mathsf{cOT}}$.

The Firewall Construction. We provide the firewalls in Fig. 21. Assuming $\mathcal{F}_{\mathsf{cOT}}$ is implemented by π_{cOT} in π_{QS}, the firewall $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ for the sender in π_{cOT} provides ER to the prover in the preprocessing phase of π_{QS}. Similarly, the firewall $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$ for the receiver in π_{cOT} provides ER to the verifier in the preprocessing phase. The coin χ is rerandomized by the firewall to prevent any exfiltration through the coin-tossing. Similarly, the commitments are also rerandomized to prevent exfiltration. Thm. 11 summarizes the RF security.

Theorem 11. *Let π_{cOT} implement $\mathcal{F}_{\mathsf{cOT}}$ in π_{QS}. Assuming Com is an additively homomorphic, binding and hiding commitment scheme, $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$ provides exfiltration resistance for a tampered receiver in π_{rOT} and $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ provides exfiltration resistance for a tampered sender of π_{cOT} then $RF_{QS\text{-}P}$ provides exfiltration resistance for the prover in π_{QS} and $\mathsf{RF}_{\mathsf{QS\text{-}V}}$ provides exfiltration resistance for the verifier in π_{QS} respectively.*

By composing Theorems 3, 10 and 11 we show that the firewalls $\mathsf{RF}_{\mathsf{QS\text{-}V}}$ and $\mathsf{RF}_{\mathsf{QS\text{-}P}}$ (Fig. 21) preserves the security of the underlying protocol π_{QS} thus proving Thm. 2.

Optimizations. Our protocol admits batching: the prover and verifier can run our protocol to verify m different circuits $(C_1, C_2, \ldots, C_m)$ with parameters $(\ell_1, \ell_2, \ldots, \ell_m)$ where ℓ_i denotes the number of input wires and multiplication gates in C_i. The parties invoke $\mathcal{F}_{\mathsf{cOT}}$ with parameter $\ell = \Sigma_{i\in[m]}\ell_i$, the combined witness $\mathbf{w}$ consists of the individual witnesses $(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$ and circuit $C(\mathbf{w}) = 1$ when $\forall i \in [m], C_i(\mathbf{w}_i) = 1$. In this batched setting, the number of public key operations for the base OTs gets amortized over m runs of the ZK protocol.

Inputs: The prover P and the verifier V hold a binary circuit C with t multiplication gates. Prover P also holds a witness assignment of wire values $\mathbf{w} \in \{0,1\}^{n+t}$ such that $C(\mathbf{w}) = 1$ and number of input wires $|\mathcal{I}_{\mathsf{wire}}|$ is n.

Preprocessing phase:
The prover and verifier knows n and t.

1. P and V invoke $\mathcal{F}_{\mathsf{cOT}}$ with $(\textsc{Initiate}, \mathsf{sid}, \mathbf{b}, \ell)$ and $(\textsc{Initiate}, \mathsf{sid}, \ell)$ respectively where $\ell = n + t + \kappa$ and $\mathbf{b} \leftarrow_R \{0,1\}^{\ell}$. $\mathcal{F}_{\mathsf{cOT}}$ returns $(\mathbf{Q}, \Delta)$ to the verifier and $\mathbf{M}$ to the prover respectively where $\mathbf{Q}, \mathbf{M} \in \{0,1\}^{\ell \times \kappa}$, $\Delta \in \{0,1\}^{\kappa}$ and $Q_i = M_i \oplus b_i \odot \Delta$ for $i \in [\ell]$.
2. The P computes pads $(A_0^*, A_1^*) \in \{0,1\}^{\kappa}$ for the batch verification as follows:
   - Set $A_0^* = (M_{n+t+1,1}, M_{n+t+2,2}, \ldots M_{n+t+\kappa,\kappa})$ where $M_{i,j}$ denotes the jth bit in the bitstring $M_i \in \{0,1\}^{\kappa}$.
   - Set $A_1^* = (b_{n+t+1}, b_{n+t+2}, \ldots, b_{\ell})$.
3. The V computes pad $B^* \in \{0,1\}^{\kappa}$ for the batch verification as follows. Set $B^* = (Q_{n+t+1,1}, Q_{n+t+2,2}, \ldots Q_{n+t+\kappa,\kappa})$ where $Q_{i,j}$ denotes the jth bit in the bitstring $Q_i \in \{0,1\}^{\kappa}$.

Online phase:
Now the circuit and witness are known by the parties.

4. *Input Wire Mapping:* For $i \in \mathcal{I}_{\mathsf{wire}}$, P sets $d_i := w_i \oplus b_i \in \{0,1\}$.
5. *Gate Computation:* For each gate $(\alpha, \beta, \gamma, T) \in C$, in a topological order:
   - If $T = \mathsf{Add}$, then prover performs nothing.
   - If $T = \mathsf{Mult}$ and this is the ith multiplication gate, the P sets $d_i = w_\alpha \cdot w_\beta \oplus b_i \in [n+1, n+t]$.

   P commits to $\mathbf{d} = \{d_i\}_{i \in [n+t]}$ as $c_{\mathbf{d}} = \mathsf{Com}(H(\mathbf{d}); \delta_{\mathbf{d}})$. P sends $c_{\mathbf{d}}$ to the verifier.
6. *Batch Verification Challenge:* P and V performs a coin tossing protocol as follows:
   - V samples $\mathsf{seed}_{\mathsf{V}} \leftarrow_R \{0,1\}^{\kappa}$ and sends $c_{\mathsf{seed}} = \mathsf{Com}(\mathsf{seed}_{\mathsf{V}}; \delta_{\mathsf{seed}})$ to P.
   - P obtains c_{seed} and samples $\mathsf{seed}_{\mathsf{P}} \leftarrow_R \{0,1\}^{\kappa}$ and sends $\mathsf{seed}_{\mathsf{P}}$ to V.
   - V opens c_{seed} by sending $(\mathsf{seed}_{\mathsf{V}}, \delta_{\mathsf{seed}})$ to P and sets $\mathsf{seed} = \mathsf{seed}_{\mathsf{P}} \oplus \mathsf{seed}_{\mathsf{V}}$.
   - P aborts if $c_{\mathsf{seed}} \neq \mathsf{Com}(\mathsf{seed}_{\mathsf{S}}; \delta_{\mathsf{seed}})$. Else, P computes challenge from the output of the coin tossing protocol, as $\chi = \mathsf{seed}_{\mathsf{S}} \oplus \mathsf{seed}_{\mathsf{R}}$.
7. *Batch Verification Response:* For the ith ($\in [t]$) multiplication gate denoted as (α, β, γ), P and V holds Q_i and M_i respectively such that $(Q_\alpha = M_\alpha \oplus b_\alpha \odot \Delta)$(same for β and γ). The wire mapping is $d_\alpha = w_\alpha \oplus b_\alpha$(same for β and γ). P computes $A_{0,i} := \mathcal{M}_\alpha \odot \mathcal{M}_\beta \in \{0,1\}^{\kappa}$ and $A_{1,i} := w_\alpha \odot M_\beta \oplus w_\beta \odot M_\alpha \odot M_\gamma$.
   - For $i \in [t]$, P computes $A_{0,i} := \mathcal{M}_\alpha \odot \mathcal{M}_\beta \in \{0,1\}^{\kappa}$ and $A_{1,i} := w_\alpha \odot M_\beta \oplus w_\beta \odot M_\alpha \odot M_\gamma$.
   - P computes $U := (\bigoplus_{i \in [t]} A_{0,i}) \odot \chi \oplus A_0^*$ and $V := (\bigoplus_{i \in [t]} A_{1,i}) \odot \chi \oplus A_1^*$. and sends (U, V) to V.

   The prover sends $(\mathbf{d}, \delta_{\mathbf{d}}, U, V)$ to the verifier as the response.
8. *Batch Verification:* The verifier performs the following:
   - Abort if $c_{\mathbf{d}} \neq \mathsf{Com}(H(\mathbf{d}); \delta_{\mathbf{d}})$.
   - For $i \in [n+t]$, set $K_i = Q_i \oplus d_i \odot \Delta$.
   - For the ith ($\in [t]$) multiplication gate denoted as (α, β, γ), set $B_i = K_\alpha \cdot K_\beta \oplus K_\gamma \odot \Delta$.
   - Compute $W = \bigoplus_{i \in [t]} B_i \oplus B^*$. Abort if $W \neq U \oplus V \odot \Delta$.

Fig. 20: Quicksilver protocol [YSWW21] π_{QS} for boolean circuit satisfiability in the firewall setting

Com is an additively homomorphic commitment scheme. $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$ and $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ provides exfiltration resistance for a tampered receiver and a tampered sender in π_{cOT}.

Preprocessing phase:
The firewall for the prover invokes the firewall $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$ (resp. $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$) to sanitize the transcript of π_{cOT} for the prover (resp. verifier).

Online phase:
Now the circuit and witness are known by the parties.

4. *Input Wire Mapping:* This step only includes local computation.
5. *Gate Computation:* Upon receiving $c_{\mathbf{d}}$ from the prover the firewall computes $\widehat{c_{\mathbf{d}}} = c_{\mathbf{d}} \cdot \mathsf{Com}(0; \widetilde{\delta_{\mathbf{d}}})$ by sampling $\widetilde{\delta_{\mathbf{d}}} \leftarrow_R \{0,1\}^\kappa$. The firewall sends $\widehat{c_{\mathbf{d}}}$ to the verifier.
6. *Batch Verification Challenge:* The steps of the coin tossing protocol are sanitised as follows:
   - Upon receiving c_{seed} from verifier the firewall computes $\widehat{c_{\mathsf{seed}}} = c_{\mathsf{seed}} \cdot \mathsf{Com}(\widetilde{\mathsf{seed}}; \widetilde{\delta_{\mathsf{seed}}})$ by sampling $\widetilde{\mathsf{seed}} \leftarrow_R \{0,1\}^\kappa$ and $\widetilde{\delta_{\mathsf{seed}}} \leftarrow_R \{0,1\}^*$. The firewall sends $\widehat{c_{\mathsf{seed}}}$ to the P.
   - Upon receiving $\mathsf{seed}_{\mathsf{P}}$ from the prover the firewall sends $\widehat{\mathsf{seed}_{\mathsf{P}}} = \mathsf{seed}_{\mathsf{P}} \oplus \widetilde{\mathsf{seed}}$ to the V.
   - Upon receiving $(\mathsf{seed}_{\mathsf{V}}, \delta_{\mathsf{seed}})$ from the verifier, the firewall sends $(\mathsf{seed}_{\mathsf{V}} \oplus \widehat{\mathsf{seed}}, \delta_{\mathsf{seed}} \oplus \widehat{\delta_{\mathsf{seed}}})$ to the prover.
7. *Batch Verification Response:* Upon receiving $(\mathbf{d}, \delta_{\mathbf{d}}, U, V)$ from the prover, the firewall sends $(\mathbf{d}, \delta_{\mathbf{d}} \oplus \widetilde{\delta_{\mathbf{d}}}, U, V)$ to the verifier as the response.
8. *Batch Verification:* This step only includes local computation.

Fig. 21: Reverse Firewalls $\mathsf{RF}_{\mathsf{QS\text{-}P}}$ (resp. $\mathsf{RF}_{\mathsf{QS\text{-}V}}$) providing exfiltration resistance for a tampered prover (resp. verifier) in π_{QS}

Acknowledgements

The work of the first author is supported in part by ERC grant 724307. The work of the second author is supported by a Google India Faculty Research Award, and Infosys Young Investigator Award, Infosys Foundation. The work of the third author is supported by NSF Awards 1931714, 1414119, and the DARPA SIEVE program.

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.

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.

BCG+19. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

BCJ21. Pascal Bemmann, Rongmao Chen, and Tibor Jager. Subversion-resilient public key encryption with practical watchdogs. LNCS, pages 627–658. Springer, Heidelberg, 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.

BM90. Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 547–557. Springer, Heidelberg, August 1990.

BMRS21. Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. LNCS, pages 92–122. Springer, Heidelberg, 2021.

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.

BPRS17. Megha Byali, Arpita Patra, Divya Ravi, and Pratik Sarkar. Fast and universally-composable oblivious transfer and commitment scheme with adaptive security. Cryptology ePrint Archive, Report 2017/1165, 2017. https://eprint.iacr.org/2017/1165.

CDN20. Suvradip Chakraborty, Stefan Dziembowski, and Jesper Buus Nielsen. Reverse firewalls for actively secure MPCs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2020, Part II*, LNCS, pages 732–762. Springer, Heidelberg, August 2020.

CDS94. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

CGPS21. Suvradip Chakraborty, Chaya Ganesh, Mahak Pancholi, and Pratik Sarkar. Reverse firewalls for adaptively secure MPC without setup. In *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 335–364. Springer, 2021.

CHY20. Rongmao Chen, Xinyi Huang, and Moti Yung. Subvert KEM to break DEM: Practical algorithm-substitution attacks on public-key encryption. In *ASIACRYPT 2020, Part II*, LNCS, pages 98–128. Springer, Heidelberg, December 2020.

CMNV22. Suvradip Chakraborty, Bernardo Magri, Jesper Buus Nielsen, and Daniele Venturi. Universally composable subversion-resilient cryptography. In *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 272–302. Springer, 2022.

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. Blazing fast OT for three-round UC OT extension. In *PKC 2020, Part II*, LNCS, pages 299–327. Springer, Heidelberg, 2020.

CSW20b. Ran Canetti, Pratik Sarkar, and Xiao Wang. Efficient and round-optimal oblivious transfer and commitment with adaptive security. In *ASIACRYPT 2020, Part III*, LNCS, pages 277–308. Springer, Heidelberg, December 2020.

DFMT20. Yevgeniy Dodis, Pooya Farshim, Sogol Mazaheri, and Stefano Tessaro. Towards defeating backdoored random oracles: Indifferentiability with bounded adaptivity. LNCS, pages 241–273. Springer, Heidelberg, March 2020.

DFP15. Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 579–598. Springer, Heidelberg, March 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.

Dia22. Benjamin E. Diamond. On the security of kos. Cryptology ePrint Archive, Paper 2022/1371, 2022. https://eprint.iacr.org/2022/1371.

DKLs18. Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997. IEEE Computer Society Press, May 2018.

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.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

FJM18. Marc Fischlin, Christian Janson, and Sogol Mazaheri. Backdoored hash functions: Immunizing HMAC and HKDF. pages 105–118, 2018.

FM18. Marc Fischlin and Sogol Mazaheri. Self-guarding cryptographic protocols against algorithm substitution attacks. pages 76–90, 2018.

GK96. Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, June 1996.

GMV20. Chaya Ganesh, Bernardo Magri, and Daniele Venturi. Cryptographic reverse firewalls for interactive proof systems. In *ICALP 2020*, LIPIcs, pages 55:1–55:16. Schloss Dagstuhl, 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.

IKNP03. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

JKO13. Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966. ACM Press, November 2013.

KOS15. Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Heidelberg, August 2015.

Mau09. Ueli M. Maurer. Unifying zero-knowledge proofs of knowledge. In Bart Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 272–286. Springer, Heidelberg, June 2009.

MR19. Daniel Masny and Peter Rindal. Endemic oblivious transfer. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 309–326. ACM Press, November 2019.

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.

NNOB12. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

NP01. Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

PSS17. Arpita Patra, Pratik Sarkar, and Ajith Suresh. Fast actively secure OT extension for short secrets. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

PVW08. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

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.

Sch90. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

WRK17. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37. ACM Press, October / November 2017.

YSWW21. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2986–3001. ACM, 2021.

YWL+20. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM CCS 20*, pages 1607–1626. ACM Press, 2020.

YWZ20. Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In *ACM CCS 20*, pages 1627–1646. ACM Press, 2020.