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

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

[1] Visa Research
[2] Indian Institute of Science, India
[3] Department of Computer Science, Boston University

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.

* The work was done while the author was at ETH Zurich. The author is supported in part by ERC grant 724307

** The author is supported by a Google India Faculty Research Award, and Infosys Young Investigator Award, Infosys Foundation.

*** The author is supported by NSF Awards 1931714, 1414119, and the DARPA SIEVE program.

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 [2] 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 [32] 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 [32] 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 [10] who constructed RFs for MPC protocols in the malicious setting. The recent work of [11] 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 [10, 11] follow roughly the same template – that of the GMW compiler [26]. 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 [24] rely on malleability of Sigma protocol, and message and key homomorphism of Pedersen commitment. The RF of [10] relies on controlled malleable non-interactive zero-knowledge proofs (NIZK)[14], and the constructions of [11] need primitives like homomorphic commitment scheme, homomorphic public-key encryption and homomorphic Sigma protocols for NP (which are secure against adaptive corruption) [9]. 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 [10, 11] 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 [28, 37, 22] based on Oblivious Transfer (OT) extension [27, 29, 34, 7, 40, 16] that only rely on cheap symmetric key operations and few public key operations. A recent line of works [38, 3] presented interactive ZK protocols for circuits in the vector OLE (Oblivious Linear Evaluation) model [5, 40, 16]. 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 [29] 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 [10, 11] 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 [38] to obtain the first efficient ZK protocol for all of NP in the RF setting. We capture our contribution by the following theorem.

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 $\mathcal{F}_{\mathsf{ZK}}$ 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.*

[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.

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 [40] 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 RF-friendly OT extension protocol can be used in a straightforward way to achieve an efficient semi-honest secure MPC using the GMW protocol. This protocol requires the parties to sample randomness for input sharing and evaluation phases. Rest of the GMW protocol is deterministic and hence would be exfiltration resistant when the parties are tampered or are semi-honest. Our RF compatible extended OTs can be used in the evaluation of multiplication gates. However, constructing a maliciously-secure MPC protocol in the GMW paradigm will require much more work. One of the reasons being the requirement of a controlled-malleable ZK protocol to ensure security against malicious adversaries in the RF setting. However, we do not know of an efficient RF-friendly instantiation, where the number of public-key operations are sub-linear in the size of the verification circuit. For other OT-based MPC protocols that rely on garbled circuits (GC), lifting our OT extension protocol to give a full-fledged RF-friendly and efficient MPC protocol seems to be more challenging. Even with our efficient OT extension protocol, one of the main bottlenecks is that we will need a *re-randomizable* GC, for which currently no efficient (in terms of public-key operations) constructions are known.

A natural extension of our work is to construct Silent OT extension family of protocols [5, 40, 16] in the RF setting. Current techniques in Silent OT extension paradigm require 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 useful in other designated-verifier settings, like silent-OT and authenticated garbling [39].

1.3 Related Work

Reverse firewalls. The work of [32] constructs RFs for a variant of the Naor-Pinkas OT protocol [33]. Their construction only provides passive security, whereas we are in the malicious setting. The work of [15] 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 [10, 11] incur $\mathsf{poly}(\kappa)$ public key operations for each OT instance. In their original paper, Mironov and Stephens-Davidowitz [32] 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 [21, 15], signature schemes [1], interactive proof systems [24], and maliciously secure MPC for both the case of static [10] and adaptive [11] corruptions. The recent work of [13] 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.

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 [10, 11] 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 the full version [12]. 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 [20]. 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 [23, 19] 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. We also refer to the full version [12] for more related works on other forms of subversion resilience based on watch-dog, self-guarding and tackling backdooring of primitives.

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 [11], 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. 3.1 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 [29]). 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 [38].

2.2 Correlated Oblivious Transfer Extension in the RF setting

We use the KOS [29] 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. 5), (implemented by base OTs) and symmetric key

[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 $(\text{INITIATE}, \mathsf{sid}, \ell)$ from sender S and receiving $(\text{INITIATE}, \mathsf{sid}, \ell)$ from receiver R, the functionality $\mathcal{F}_{\mathsf{cOT}}$ interacts as follows:

- Sample $\mathbf{b} = (b_1, \dots, b_\ell)$ where $b_j \in \{0,1\}$ for $j \in [\ell]$.
- If S is corrupted receive $\mathbf{s} \in \{0,1\}^\kappa$ from the sender. Sample $\mathbf{Q} \leftarrow_R \{0,1\}^{\ell \times \kappa}$. 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 $(\text{GUESS}, \mathsf{sid}, \{\mathsf{ind}_i\}_{i \in [c]}, \{s'_i\}_{i \in [c]})$: $\mathcal{F}_{\mathsf{rOT}}$ 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 $(\text{UNDETECTED}, \mathsf{sid})$ 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 $(\mathsf{sent}, \mathsf{sid}, \mathbf{M}, \mathbf{b})$ 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.

If a corrupt sender (resp. receiver) sends $\perp$ to the $\mathcal{F}_{\mathsf{cOT}}$ then $\mathcal{F}_{\mathsf{cOT}}$ delivers the output of the corrupt sender (resp. receiver) to the corrupt sender (resp. receiver) and aborts.

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

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:

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

$\mathsf{R}_{\mathsf{Ext}}$ 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 [18]. 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 [35, 8] 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 [6, 30, 7] 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 $\mathbf{D}$ 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 as $\mathcal{F}_{\mathsf{RO}}(0, \mathsf{coin})$[6].

- **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

[6] The $\mathcal{F}_{\mathsf{RO}}$ functionality is parametrized by 0 so that we can reuse the same functionality later for a different input/output pair by changing the parameter to 1

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 $\mathcal{F}_{\mathsf{RO}}(1, \mathsf{seed})$, 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 $\widehat{c_{\mathsf{seed}}} = c_{\mathsf{seed}} \cdot \mathsf{Com}(\widetilde{\mathsf{seed}}; \widetilde{\delta_{\mathsf{seed}}})$ 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. 3 and the firewall in Fig. 4. Our correlated OT with leakage is weaker than correlated OT of [38] 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 [38]. 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 [35, 30, 8, 7] fail to give π_{rOT} in the presence of functionality maintaining tampering. The OT protocol of [32] 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 [4] in the plain model. For the sake of completeness, we first recall the protocol.

Protocol of [4]. 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 [4] 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 $((\mathsf{pk}_0, \mathsf{pk}_1, \mathbb{G}, \mathbb{Z}_q) : \exists \mathsf{sk} \in \mathbb{Z}_q, b \in \{0,1\} \text{ s.t. } (\mathsf{pk}_0 = g^{\mathsf{sk}} \vee \mathsf{pk}_1 = g^{\mathsf{sk}}))$ 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 $((\mathsf{pk}_0, \mathsf{pk}_1, \mathbb{G}, \mathbb{Z}_q) : \exists w \in \mathbb{Z}_q, b \in \{0,1\} \text{ s.t. } (\mathsf{pk}_0 = g^w \vee \mathsf{pk}_1 = g^w))$ 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 $((\mathsf{pk}_0 \cdot g^{\widetilde{p}}, \mathsf{pk}_1 \cdot g^{-\widetilde{p}}, \mathbb{G}, \mathbb{Z}_q) : \exists w \in \mathbb{Z}_q, b \in \{0,1\} \text{ s.t. } (\mathsf{pk}_0 \cdot g^{\widetilde{p}} = g^w \vee \mathsf{pk}_1 \cdot g^{-\widetilde{p}} = g^w))$. If $\widetilde{\rho} = 1$ the new statement is $((\mathsf{pk}_0 \cdot g^{\widetilde{p}}, \mathsf{pk}_1 \cdot g^{-\widetilde{p}}, \mathbb{G}, \mathbb{Z}_q) : \exists w \in \mathbb{Z}_q, b \in \{0,1\} \text{ s.t. } (\mathsf{pk}_0 \cdot g^{\widetilde{p}} = g^w \vee \mathsf{pk}_1 \cdot g^{-\widetilde{p}} = g^w))$. 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. 6 and 7. 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 [10, 11] 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 [36]. 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 [24] 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 [24].

2.5 Efficient Zero-Knowledge in the RF setting

The recent works of [38, 3] present interactive ZK protocols for circuits in the vector OLE model [5, 40]. We focus on the work of Quicksilver [38] 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 $A_{i,1}$:

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

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 $\varDelta$ 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 $(C_1, C_2, \dots, C_m)$ with parameters $(\ell_1, \ell_2, \dots, \ell_m)$ 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 $L = \varSigma_{i \in [m]} \ell_i$. 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 $\{1, \dots, n\}$ 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 $\mathbf{a} \odot \mathbf{b} = (a_1 \cdot b_1, \ldots, a_n \cdot b_n)$.

Commitment Schemes: We define a non-interactive commitment scheme Com as a tuple of two algorithms (Gen, Com) such that it satisfies the properties of computational binding, computational hiding. Additionally, we require Com to be additively homomorphic over the 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: $\mathsf{Com}(\mathsf{pp}, m; r) \cdot \mathsf{Com}(\mathsf{pp}, m'; r') = \mathsf{Com}(\mathsf{pp}, m + m'; r + r')$, where pp are the public parameters generated by Gen. For our protocols we require additively homomorphic commitments over $\mathbb{Z}_q$ and $\mathbb{G}$ message spaces. We use Pedersen and Elgamal commitments respectively for this purpose. More details can be found in the full version [12].

3.1 Cryptographic Reverse Firewalls

In this section we recall the basic definitions of reverse firewalls following [32, 10, 11]. We focus on the setting of two parties.

Notation. Let $\mathit{\Pi}$ 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 $\mathit{\Pi}_{\mathsf{RF} \circ P}$ (resp. $\mathit{\Pi}_{\overline{P}}$) to represent the protocol $\mathit{\Pi}$ 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 1 (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* $\mathit{\Pi}$ *that satisfies some functionality requirements* $\mathcal{F}$*, we say that a reverse firewall* RF maintains functionality $\mathcal{F}$ *for a party* P *in protocol* $\mathit{\Pi}$ *if* $\mathit{\Pi}_{\mathsf{RF}^k \circ P}$ *also satisfies* $\mathcal{F}$*, for any polynomially bounded* $k \geq 1$*.*

Definition 2 (Security preservation). *A reverse firewall* weakly preserves security $\mathcal{S}$ for party P in protocol $\mathit{\Pi}$ *if protocol* $\mathit{\Pi}$ *satisfies* $\mathcal{S}$*, and for any polynomial-time algorithm* $\overline{P}$ *such that* $\mathit{\Pi}_{\overline{P}}$ *satisfies* $\mathcal{F}$*, the protocol* $\mathit{\Pi}_{\mathsf{RF} \circ \overline{P}}$ *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.)*

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.)*

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 3 (Exfiltration resistance). *A reverse firewall is* weak exfiltration resistant for party P_1 against party P_2 in protocol Π *satisfying functionality $\mathcal{F}$ if no PPT adversary $\mathcal{A}_{\mathsf{ER}}$ 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* weak 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.

```
$\underline{\mathsf{LEAK}(\Pi, P_1, P_2, \mathsf{RF}, \kappa)}$
$(\overline{P_1}, \overline{P_2}, I) \leftarrow \mathcal{A}_{\mathsf{ER}}(1^\kappa)$
$b \xleftarrow{\$} \{0,1\}$;
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 [11] that intuitively, requires that the behavior of $\mathsf{RF}\circ P$ is identical to the behavior of P if P is the honest implementation.

Throughout the paper we refer to weak exfiltration resistance as exfiltration resistance. We will also use the following result established in [11]. It basically states that exfiltration resistance implies security preservation for protocols satisfying simulation-based definition of security.

Theorem 3 ([11] 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 2.*

4 Correlated OT Extension in the Firewall Setting

We describe our revised cOT extension protocol in Fig. 3 and the corresponding firewall can be found in Fig. 4. High level overview can be found in Sec. 2.2. We show security of our protocol by proving Thm. 4 in the full version [12].

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.*

We show that our protocol provides weak exfiltration resistance against tampering of honest parties by proving Thm. 5 as follows.

Theorem 5. *Assuming* Com *is an additively homomorphic, binding and hiding commitment scheme, and $\mathsf{RF}_{\mathsf{rOT\text{-}R}}$ provides weak exfiltration resistance for the receiver (of base OT) in π_{rOT} then $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ (Fig. 4) provides weak exfiltration resistance for a tampered sender of π_{cOT}. Similarly if $\mathsf{RF}_{\mathsf{rOT\text{-}S}}$ provides weak exfiltration resistance for the sender (of base OT) in π_{rOT} then $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$ (Fig. 4) provides weak exfiltration resistance for a tampered receiver in π_{cOT}.*

Proof. We argue weak 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 $\mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, 0, \mathsf{coin})$, $\mathsf{PRG}(k_{1,0})$ 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:

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

  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 $s_i == 1$.

- **Private Inputs:** R and S do not possess any private inputs.
- **Primitives:** Pseudorandom Generators $\mathsf{PRG} : \{0,1\}^\kappa \to \{0,1\}^{m+\kappa}$, $H : \{0,1\}^{(m+\kappa)\times\kappa} \to \{0,1\}^\kappa$ is a Collision Resistant Hash function and $\mathsf{Com} : \{0,1\}^\kappa \to \{0,1\}^\kappa$ is a string commitment scheme. $\mathcal{F}_{\mathsf{RO}}$ is a random oracle functionality such that $\mathcal{F}_{\mathsf{RO}} : \{0,1\}^\kappa \times \{0\} \to \{0,1\}^{m+\kappa}$ and $\mathcal{F}_{\mathsf{RO}} : \{0,1\}^\kappa \times \{1\} \to \mathbb{F}^{m+\kappa}$.
- **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 $\mathbf{D}^1 = \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, 0, \mathsf{coin})$. 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 $\chi = \{\chi_1, \ldots, \chi_{m+\kappa}\} = \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, 1, \mathsf{seed}_\mathsf{S} + \mathsf{seed}_\mathsf{R})$.
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 $\mathbf{D}^1 \neq \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, 0, \mathsf{coin}_\mathsf{R} \oplus \mathsf{coin}_\mathsf{S})$. 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 $\chi = \mathcal{F}_{\mathsf{RO}}(\mathsf{sid}, 1, \mathsf{seed})$ 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 $(\mathbf{s}, \{\mathbf{Q}_j\}_{j\in[m]})$ as the output. R sets $(\mathbf{r}, \{\mathbf{M}_j\}_{j\in[m]})$ as the output.

Fig. 3: 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 $\mathsf{coins}_{\mathsf{S}}$ from the sender, the firewall sends $\widehat{\mathsf{coins}_{\mathsf{S}}} = \mathsf{coins}_{\mathsf{S}} + \widetilde{\mathsf{coin}}$ 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 $\mathsf{seed}_{\mathsf{R}}$, the firewall sends $\widetilde{\mathsf{seed}_{\mathsf{R}}} = \mathsf{seed}_{\mathsf{R}} + \widetilde{\mathsf{seed}}$ to the sender S.
   - When S sends $(\mathsf{seeds}_{\mathsf{S}}, \delta_{\mathsf{seed}})$, the firewall sends $(\widehat{\mathsf{seeds}_{\mathsf{S}}}, \widehat{\delta_{\mathsf{seed}}}) = (\mathsf{seeds}_{\mathsf{S}} + \widetilde{\mathsf{seed}}, \delta_{\mathsf{seed}} + \widetilde{\delta_{\mathsf{seed}}})$ 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 $(\mathsf{coin}_{\mathsf{R}}, \delta_{\mathsf{coin}})$, the firewall sends $(\mathsf{coin}_{\mathsf{R}} + \widetilde{\mathsf{coin}}, \delta_{\mathsf{coin}} + \widetilde{\delta_{\mathsf{coin}}})$ to the sender.

Fig. 4: Sender's (resp. Receiver's) Firewall $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ (resp. $\mathsf{RF}_{\mathsf{cOT\text{-}R}}$) in π_{cOT}

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. 4) 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. 5) for base OT protocol. Our protocol π_{rOT} can be found in the full version [12]. Detailed overview can be found in Sec. 2.3. We show simulation based security of π_{rOT} by proving Thm. 6 in the full version [12]. We implement the ZK protocol in Fig. 9 and WI protocol in Fig. 10 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. 5: 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, $\pi_{\mathsf{ZK}}^{\mathsf{DL}}$ implement $\mathcal{F}_{\mathsf{ZK}}$ functionality for the Discrete Log relation $\mathcal{R}_{\mathsf{DL}}$, $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ 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.*

We provide the reverse firewall $\mathsf{RF}_{\mathsf{rOT}}$ for protocol π_{rOT} in the full version [12]. 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 weak exfiltration resistance for the tampered parties in $\pi_{\mathsf{ZK}}^{\mathsf{DL}}$ and $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ respectively, then the above firewall $\mathsf{RF}_{\mathsf{rOT}}$ provides weak exfiltration resistance for a tampered sender against a receiver, and for a tampered receiver against a sender.*

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

$\mathsf{Com}_{\mathbb{G}}$ and Com_q are commitments for group elements and field elements respectively. $\pi_{\mathsf{ZK}}^{\mathsf{DL}}$ is a ZK proof for the statement $(x, \mathbb{G}, \mathbb{Z}_q)$ corresponding to relation $\mathcal{R}_{\mathsf{DL}} = (\exists w \in \mathbb{Z}_q : x = g^w)$. $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ 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 $c_i^{\mathsf{S}} = \mathsf{Com}_q(p_i; d_i^{\mathsf{S}})$.

   The sender 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 sender and the receiver run $\pi_{\mathsf{ZK}}^{\mathsf{DL}}$ 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 $c_{i,0}^{\mathsf{R}} = \mathsf{Com}_q(v_{i,0}; d_{i,0}^{\mathsf{R}})$ and $c_{i,1}^{\mathsf{R}} = \mathsf{Com}_q(v_{i,1}; d_{i,1}^{\mathsf{R}})$.

   The 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 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 $\{\mathsf{sk}_i, b_i\}_{i \in [\ell]}$.
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 $(\boldsymbol{\rho}, \{R_{i,0}, R_{i,1}, p_i, d_i^{\mathsf{S}}\}_{i \in [\ell]})$ 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 $c_i^{\mathsf{S}} \neq \mathsf{Com}_q(p_i; d_i^{\mathsf{S}})$.
   - 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 $(\mathbf{C}_0^{\mathsf{R}}, \mathbf{C}_1^{\mathsf{R}})$ by sending $\{v_{i,0}, d_{i,0}^{\mathsf{R}}, v_{i,1}, d_{i,1}^{\mathsf{R}}\}_{i \in [\ell]}$ to sender.

Fig. 6: 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. 7: Protocol π_{rOT} implementing $\mathcal{F}_{\mathsf{rOT}}$

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 [24] 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. We demonstrate that the sigma protocol for discrete log is malleable in Fig. 8.

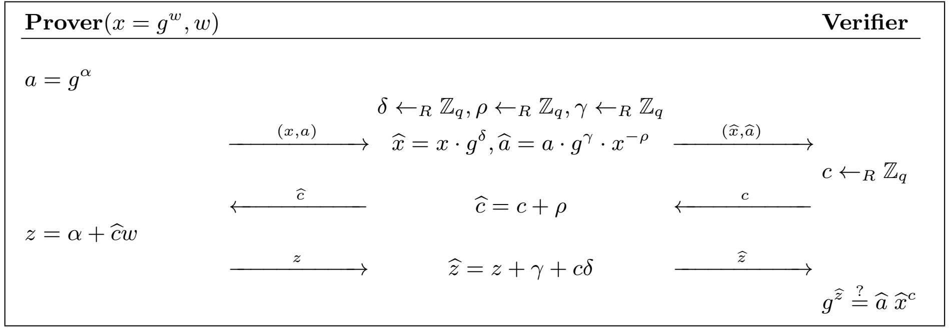

Prover$(x = g^w, w)$
Verifier
$a = g^\alpha$
$\delta \leftarrow_R \mathbb{Z}_q, \rho \leftarrow_R \mathbb{Z}_q, \gamma \leftarrow_R \mathbb{Z}_q$
(x,a)
$\widehat{x} = x \cdot g^\delta, \widehat{a} = a \cdot g^\gamma \cdot x^{-\rho}$
$(\widehat{x},\widehat{a})$
$c \leftarrow_R \mathbb{Z}_q$
$\widehat{c}$
$\widehat{c} = c + \rho$
c
$z = \alpha + \widehat{c}w$
z
$\widehat{z} = z + \gamma + c\delta$
$\widehat{z}$
$g^{\widehat{z}} \stackrel{?}{=} \widehat{a}\,\widehat{x}^c$

Fig. 8: Fully Malleable Sigma protocol for discrete log

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

Definition 4 (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:*

(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 c 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 $\mathsf{P}_1(\widehat{x}, \widehat{w})$, 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 $\mathsf{V}(x, (a, \widehat{c}, z)) = 1$, where $\widehat{c} \leftarrow_R \mathsf{MaulCh}(c; \rho)$, the following holds :*

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

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

Lemma 1. *The Sigma protocol for Discrete Log is fully malleable as per Definition 4. The construction is shown in Fig 8.*

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

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

- *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 $g^z = a x^{-\widehat{c}}$, where $\widehat{c} = c + \rho$, the following holds:

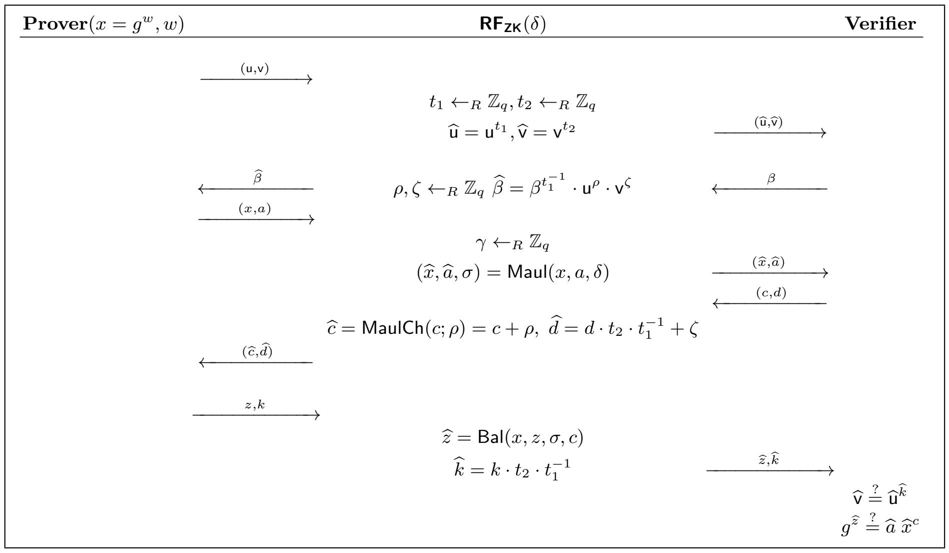
$$\widehat{a}\widehat{x}^{-c} = a g^{\gamma} x^{-\rho} \widehat{x}^{-c} = a g^{\gamma} x^{-c-\rho} g^{-\delta c} = a g^{\gamma} x^{-\widehat{c}} g^{-\delta c} = g^z g^{\gamma} g^{-\delta c} = g^{\widehat{z}}$$

□

We note that Maul and Bal easily generalize to the unifying Sigma protocol for proving knowledge of preimage of a homomorphism [31]. 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 [25] allow to compile a Sigma protocol into a zero knowledge protocol. We recall the ZKPoK protocol $\pi_{\mathsf{ZK}}^{\mathsf{DL}}$ for the discrete logarithm problem in the full version [12] and we provide an RF for it in Fig. 9 and prove Thm. 8.

Theorem 8. *Let Σ be a fully malleable unique-response Sigma protocol for $\mathcal{R}$ as in Def 4. The RF $\mathsf{RF}_{\mathsf{ZK}}$ in Fig. 9 is functionality-maintaining, weakly ZK preserving and weak exfiltration resistant for the ZK protocol π_{ZK}^{DL} of discrete log.*



Prover$(x = g^w, w)$
$\mathsf{RF}_{\mathsf{ZK}}(\delta)$
Verifier
(u,v)
$t_1 \leftarrow_R \mathbb{Z}_q, t_2 \leftarrow_R \mathbb{Z}_q$
$\widehat{\mathsf{u}} = \mathsf{u}^{t_1}, \widehat{\mathsf{v}} = \mathsf{v}^{t_2}$
$(\widehat{\mathsf{u}},\widehat{\mathsf{v}})$
$\widehat{\beta}$
$\rho, \zeta \leftarrow_R \mathbb{Z}_q \ \widehat{\beta} = \beta^{t_1^{-1}} \cdot \mathsf{u}^{\rho} \cdot \mathsf{v}^{\zeta}$
β
(x,a)
$\gamma \leftarrow_R \mathbb{Z}_q$
$(\widehat{x}, \widehat{a}, \sigma) = \mathsf{Maul}(x, a, \delta)$
$(\widehat{x},\widehat{a})$
(c,d)
$\widehat{c} = \mathsf{MaulCh}(c; \rho) = c + \rho, \ \widehat{d} = d \cdot t_2 \cdot t_1^{-1} + \zeta$
$(\widehat{c},\widehat{d})$
z,k
$\widehat{z} = \mathsf{Bal}(x, z, \sigma, c)$
$\widehat{k} = k \cdot t_2 \cdot t_1^{-1}$
$\widehat{z},\widehat{k}$
$\widehat{\mathsf{v}} \overset{?}{=} \widehat{\mathsf{u}}^{\widehat{k}}$
$g^{\widehat{z}} \overset{?}{=} \widehat{a}\,\widehat{x}^{c}$

Fig. 9: Reverse Firewall $\mathsf{RF}_{\mathsf{ZK}}$ for ZK compiled Sigma protocol

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 [17]. We describe the protocol $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ in Fig. 10. $\pi_{\mathsf{WI}}^{\mathsf{OR}}$ 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 [24] 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 present the WI protocol for OR composition the full version [12]. We show the RF for the OR composition in Fig. 10 and demonstrate that it provides ER by proving Thm. 9.

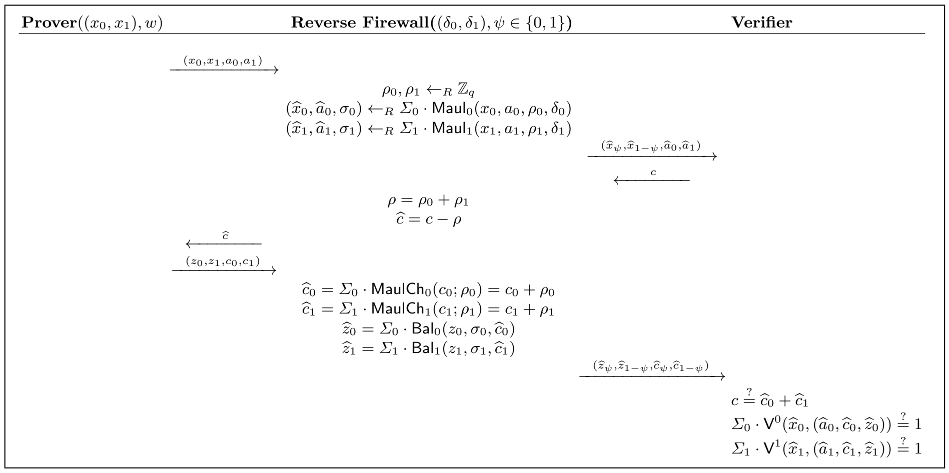

Prover$((x_0, x_1), w)$
Reverse Firewall$((\delta_0, \delta_1), \psi \in \{0, 1\})$
Verifier
$\xrightarrow{(x_0, x_1, a_0, a_1)}$
$\rho_0, \rho_1 \leftarrow_R \mathbb{Z}_q$
$(\widehat{x}_0, \widehat{a}_0, \sigma_0) \leftarrow_R \Sigma_0 \cdot \mathsf{Maul}_0(x_0, a_0, \rho_0, \delta_0)$
$(\widehat{x}_1, \widehat{a}_1, \sigma_1) \leftarrow_R \Sigma_1 \cdot \mathsf{Maul}_1(x_1, a_1, \rho_1, \delta_1)$
$\xrightarrow{(\widehat{x}_\psi, \widehat{x}_{1-\psi}, \widehat{a}_0, \widehat{a}_1)}$
$\xleftarrow{c}$
$\rho = \rho_0 + \rho_1$
$\widehat{c} = c - \rho$
$\xleftarrow{\widehat{c}}$
$\xrightarrow{(z_0, z_1, c_0, c_1)}$
$\widehat{c}_0 = \Sigma_0 \cdot \mathsf{MaulCh}_0(c_0; \rho_0) = c_0 + \rho_0$
$\widehat{c}_1 = \Sigma_1 \cdot \mathsf{MaulCh}_1(c_1; \rho_1) = c_1 + \rho_1$
$\widehat{z}_0 = \Sigma_0 \cdot \mathsf{Bal}_0(z_0, \sigma_0, \widehat{c}_0)$
$\widehat{z}_1 = \Sigma_1 \cdot \mathsf{Bal}_1(z_1, \sigma_1, \widehat{c}_1)$
$\xrightarrow{(\widehat{z}_\psi, \widehat{z}_{1-\psi}, \widehat{c}_\psi, \widehat{c}_{1-\psi})}$
$c \overset{?}{=} \widehat{c}_0 + \widehat{c}_1$
$\Sigma_0 \cdot \mathsf{V}^0(\widehat{x}_0, (\widehat{a}_0, \widehat{c}_0, \widehat{z}_0)) \overset{?}{=} 1$
$\Sigma_1 \cdot \mathsf{V}^1(\widehat{x}_1, (\widehat{a}_1, \widehat{c}_1, \widehat{z}_1)) \overset{?}{=} 1$

Fig. 10: $\mathsf{RF}_{\mathsf{WI}}$: 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)

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. 10 preserves completeness, is weakly HVZK/WI preserving and weak exfiltration resistant for $\pi_{\mathsf{WI}}^{\mathsf{OR}}$.*

7 Quicksilver with Reverse Firewall

We present a variant of Quicksilver [38] in the firewall setting, π_{QS}, in the full version [12]. 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 Section 2.5. Security of π_{QS} is summarized in Thm. 10. More details can be found in the full version [12].

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

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. 11: 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}

The Firewall Construction. We provide the firewalls in Fig. 11. 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 weak exfiltration resistance for a tampered receiver in π_{rOT} and* $\mathsf{RF}_{\mathsf{cOT\text{-}S}}$ *provides weak exfiltration resistance for a tampered sender of π_{cOT} then $RF_{QS\text{-}P}$ provides weak exfiltration resistance for the prover in π_{QS} and* $\mathsf{RF}_{\mathsf{QS\text{-}V}}$ *provides weak 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. 11) 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.

References

1. Ateniese, G., Magri, B., Venturi, D.: Subversion-resilient signature schemes. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 364–375. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813635
2. Ball, J., Borger, J., Greenwald, G., et al.: Revealed: how us and uk spy agencies defeat internet privacy and security. Know Your Neighborhood (2013)
3. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 92–122. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_4
4. Bellare, M., Micali, S.: Non-interactive oblivious transfer and applications. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 547–557. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0_48
5. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 291–308. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3354255

6. Byali, M., Patra, A., Ravi, D., Sarkar, P.: 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
7. Canetti, R., Sarkar, P., Wang, X.: Blazing fast OT for three-round UC OT extension. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 299–327. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45388-6_11
8. Canetti, R., Sarkar, P., Wang, X.: Efficient and round-optimal oblivious transfer and commitment with adaptive security. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 277–308. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64840-4_10
9. Canetti, R., Sarkar, P., Wang, X.: Triply adaptive UC NIZK. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13792, pp. 466–495. Springer (2022)
10. Chakraborty, S., Dziembowski, S., Nielsen, J.B.: Reverse firewalls for actively secure MPCs. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 732–762. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_26
11. Chakraborty, S., Ganesh, C., Pancholi, M., Sarkar, P.: 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. Lecture Notes in Computer Science, vol. 13091, pp. 335–364. Springer (2021)
12. Chakraborty, S., Ganesh, C., Sarkar, P.: Reverse firewalls for oblivious transfer extension and applications to zero-knowledge. IACR Cryptol. ePrint Arch. p. 1535 (2022), https://eprint.iacr.org/2022/1535, https://eprint.iacr.org/2022/1535
13. Chakraborty, S., Magri, B., Nielsen, J.B., Venturi, D.: 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. Lecture Notes in Computer Science, vol. 13275, pp. 272–302. Springer (2022), https://doi.org/10.1007/978-3-031-06944-4_10
14. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 281–300. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_18
15. Chen, R., Mu, Y., Yang, G., Susilo, W., Guo, F., Zhang, M.: Cryptographic reverse firewall via malleable smooth projective hash functions. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 844–876. Springer, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_31
16. Couteau, G., Rindal, P., Raghuraman, S.: Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 502–534. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84252-9_17
17. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO'94.

LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48658-5_19
18. Diamond, B.E.: On the security of kos. Cryptology ePrint Archive, Paper 2022/1371 (2022), https://eprint.iacr.org/2022/1371, https://eprint.iacr.org/2022/1371
19. Dodis, Y., Farshim, P., Mazaheri, S., Tessaro, S.: Towards defeating backdoored random oracles: Indifferentiability with bounded adaptivity. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part III. LNCS, vol. 12552, pp. 241–273. Springer, Heidelberg (Nov 2020). https://doi.org/10.1007/978-3-030-64381-2_9
20. Dodis, Y., Ganesh, C., Golovnev, A., Juels, A., Ristenpart, T.: A formal treatment of backdoored pseudorandom generators. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 101–126. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_5
21. Dodis, Y., Mironov, I., Stephens-Davidowitz, N.: Message transmission with reverse firewalls—secure communication on corrupted machines. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 341–372. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_13
22. Doerner, J., Kondi, Y., Lee, E., shelat, a.: Secure two-party threshold ECDSA from ECDSA assumptions. In: 2018 IEEE Symposium on Security and Privacy. pp. 980–997. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00036
23. Fischlin, M., Janson, C., Mazaheri, S.: Backdoored hash functions: Immunizing HMAC and HKDF. In: Chong, S., Delaune, S. (eds.) CSF 2018 Computer Security Foundations Symposium. pp. 105–118. IEEE Computer Society Press (2018). https://doi.org/10.1109/CSF.2018.00015
24. Ganesh, C., Magri, B., Venturi, D.: Cryptographic reverse firewalls for interactive proof systems. In: Czumaj, A., Dawar, A., Merelli, E. (eds.) ICALP 2020. LIPIcs, vol. 168, pp. 55:1–55:16. Schloss Dagstuhl (Jul 2020). https://doi.org/10.4230/LIPIcs.ICALP.2020.55
25. Goldreich, O., Kahan, A.: How to construct constant-round zero-knowledge proof systems for NP. Journal of Cryptology **9**(3), 167–190 (Jun 1996)
26. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987). https://doi.org/10.1145/28395.28420
27. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_9
28. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 955–966. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516662
29. Keller, M., Orsini, E., Scholl, P.: Actively secure OT extension with optimal overhead. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 724–741. Springer, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-47989-6_35
30. Masny, D., Rindal, P.: Endemic oblivious transfer. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 309–326. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3354210

31. Maurer, U.M.: Unifying zero-knowledge proofs of knowledge. In: Preneel, B. (ed.) AFRICACRYPT 09. LNCS, vol. 5580, pp. 272–286. Springer, Heidelberg (Jun 2009)
32. Mironov, I., Stephens-Davidowitz, N.: Cryptographic reverse firewalls. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 657–686. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6_22
33. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Kosaraju, S.R. (ed.) 12th SODA. pp. 448–457. ACM-SIAM (Jan 2001)
34. Patra, A., Sarkar, P., Suresh, A.: 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)
35. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (Aug 2008). https://doi.org/10.1007/978-3-540-85174-5_31
36. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0_22
37. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 21–37. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134053
38. Yang, K., Sarkar, P., Weng, C., Wang, X.: Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021. pp. 2986–3001. ACM (2021)
39. Yang, K., Wang, X., Zhang, J.: More efficient MPC from improved triple generation and authenticated garbling. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1627–1646. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417285
40. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1607–1626. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417276