Protection Against Subversion Corruptions via Reverse Firewalls in the Plain Universal Composability Framework

Paula Arnold[1], Sebastian Berndt[2], Jörn Müller-Quade[3,4], and Astrid Ottenhues[3,4]

[1] University of Luebeck
p.arnold@uni-luebeck.de
[2] Technische Hochschule Lübeck
sebastian.berndt@th-luebeck.de
[3] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[4] KASTEL Security Research Labs, Karlsruhe, Germany
{mueller-quade, ottenhues}@kit.edu

Abstract. While many modern cryptographic primitives have stood the test of time, attackers started to expand beyond classic cryptanalysis by targeting implementations. Subversion attacks, where the attacker replaces the implementation of the cryptographic primitive to leak sensitive information about the user during a protocol execution, are among the most dangerous of such attacks. The revelations of Snowden have shown that these attacks are deployed by intelligence services. A very promising countermeasure uses *cryptographic reverse firewalls* that actively remove the covert channel leaking the secret. Chakraborty et al. (EUROCRYPT'22) presented the first model of such firewalls in the universal composability (UC) framework. However, using such a firewall also provides a possible new target for the attacker and in the case that an honest party uses a corrupted firewall, they were not able to prove any security guarantees. Furthermore, their model is quite complex and does not fit into the *plain* UC model as they restrict the environment. Hence, the authors needed to reprove fundamental theorems such as the composition theorem as well as the security of the underlying protocol. In this paper, we consider a slightly different model of subversion attacks that replace the used randomness, inspired by Dodis et al. (CRYPTO'16), that captures all known subversion attacks. Considering these realistic attacks allows us to use *existing* UC-secure protocols without the need to reprove their security. We also introduce additional notions of firewall properties, allowing us to reason about corrupted firewalls while maintaining strong security guarantees. To show the versatility of our model, we apply it to commitments and oblivious transfer. This demonstrates the usefulness of our plain UC model, as the only known previous subversion-resilient OT, recently provided by Chakraborty et al. (ASIACRYPT'24), is much more complicated and involved, and was also in the non-plain UC model.

Keywords: Subversion Resilience · Universal Composability · OT

1 Introduction

Nowadays, many cryptographic primitives such as AES or ECDH have stood the test of time and have been in use for decades without any major security loss due to improved attacks via cryptanalysis. However, already in the late seventies, Simmons noticed the danger of cryptographic implementations embedding a *covert channel* that can leak sensitive information only visible to the attacker [36]. Such attacks were later studied more formally under the name of *kleptography* by Young and Yung [38, 39]. While the existence of such attacks was thus clearly demonstrated on a theoretical level, no real-world attack was known so far. This drastically changed when the issues surrounding the Dual Elliptic Curve Deterministic Random Bit Generator (Dual_EC_DRBG) came to light that incorporated a backdoor provided by the NSA [19]. Furthermore, later revelations due to Snowden exposed the existence of NSA's *Project Bullrun*. Some goals of this project were to "*Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets*" and to "*Influence policies, standards and specification for commercial public key technologies*" [33]. These revelations led to an increased alertness by cryptographic researchers that quickly started to develop possible countermeasures against such *subversion attacks* [5, 4]. While these countermeasures have not yet found their way into real-world systems, the danger of subversion attacks is actively discussed in real-world scenarios. For example, a specific countermeasure[5] was proposed in the standardisation process of post-quantum cryptography to prevent such attacks [23], but was ultimately rejected.

Unfortunately, there are subversion attacks that are *completely undetectable* in a black-box setting [6] and all countermeasures thus need to use a non-black-box approach. One of the most widely studied countermeasures is the so-called *reverse firewall* [32, 24] that aims to actively remove the covert channel from a subverted implementation, often via means of rerandomisation. In general, such firewalls are devices placed between the different parties of a protocol, e.g., by being directly incorporated into routers. A cautious user wanting to protect their data against subverted implementations thus might decide to install such a firewall. While the use of a suitably chosen firewall can prevent subversion attacks, they can also enlarge the attack surface, as the firewall itself is now a new corruption target. Clearly, a cautious user would only use such a firewall if the additional attack surface is limited, i.e., if the firewall cannot lower the security guarantees significantly.

While different protocols and corresponding reverse firewalls have been presented in the literature, Chakraborty et al. [18] noticed that their use of game-based security notions could lead to problems when considering the security of concurrent runs of the underlying protocol. Hence, realistic attackers could still leak sensitive information via the covert channel by starting multiple concurrent sessions with the subverted protocol, even in the presence of these sophisticated countermeasures. To remedy this situation, they proposed an extension of the

[5] This countermeasure has also been called the *hash of shame*.

Universal Composability (UC) model by Canetti [12, 10] to capture subversion attacks and countermeasures built on reverse firewalls. Based on this, they constructed a secure protocol for commitments and a reverse firewall for this protocol as well as a completeness theorem for maliciously secure MPC in their model by instantiating the Goldreich-Micali-Wigderson (GMW) compiler [26]. While these results are the first results guaranteeing security against subversion attacks in the context of concurrent runs, they come with a drawback. The original reverse firewalls guaranteed a high degree of *transparency*, i.e., neither the sender nor the receiver needed to be aware of the existence of the reverse firewall [32, 24]. Transparency allows for an elegant argument if the firewall itself is corrupted but the protocol is secure against malicious attackers: For each individual party, we can treat the corrupted firewall to be "external", i.e., to be part of a different, maliciously corrupted party. Hence, the security of the original protocol directly implies security against a corrupted firewall. In contrast, in the model presented by Chakraborty et al., the sender is actively communicating with the reverse firewall via a *feedback channel* and thus needs to be aware of the existence of the firewall. Hence, the firewall cannot be transparent for the sender. At first glance, this looks like a minor annoyance without much effect. However, this has far-reaching consequences for the guarantees achievable by their model, as a corrupted firewall can now drastically reduce the security guarantees. Consider as an example that an honest, non-subverted implementation is used. In the model of Chakraborty et al., albeit giving this corruption case a different name, a maliciously corrupted firewall leads to the effect that the complete *combined party* (i.e., the party together with the firewall) will be treated as maliciously corrupted. But, as the implementation is honest, the firewall should not be able to extract any sensitive information from interacting with an honest party. Hence, there is a clear gap between the security guarantees provided by the model of Chakraborty et al. and the guarantees achieved in the real world. Note that the attacker model of Chakraborty et al. allows for very strong attacks, including attacks where the corrupted implementation completely deviates from its usual behaviour, e.g., by changing the message of a message transmission protocol. However, as discussed by Dodis et al. [24], protecting message transmission protocols against such functionality-changing attacks is very hard (or impossible, depending on the model). Hence, Chakraborty et al. need to focus on functionalities with *random* inputs or trust the firewall to not change the input. Furthermore, due to the complex generalisation of the already complex UC model, Chakraborty et al. need to reprove several fundamental results such as the *composition theorem*. Finally, as the authors use a generalisation of the UC model, to convert known UC-secure protocols to their model they need to essentially *reprove* the security of the protocol.

Due to space limitations, some of the results are only sketched in this version. We refer the reader to the full version [2] available at https://eprint.iacr.org/2023/1951 for more information.

1.1 Contributions

In this paper, we present an alternative model to the one of Chakraborty et al. that is in plain UC which allows us to precisely capture and minimise the attack vectors available to a corrupted firewall. This alternative model follows the threat model of Dodis et al. [24] by only considering corruptions preserving the underlying functionality. This allows us to also protect functionalities with non-random inputs, to reuse existing security proofs, and guarantee security in the presence of corrupted firewalls. We show the versatility of our approach by presenting UC-secure subversion resilient protocols for commitments and OT. In more detail, our contributions are:

- We present the first subversion corruption model in plain UC which allows us to use all of the existing tools such as the composition theorem and to also use existing UC-secure primitives.
- We show that it is possible to avoid subversion attacks already in real world protocols (i.e., $\Pi \geq_{\mathrm{UC}} \Pi'$), such that there is no need to have the subversion corruption in the ideal world, while still having the security properties of subversion resilience (i.e, $\Pi \geq_{\mathrm{UC}} \mathcal{F}$).
- To simplify the steps necessary when proving the UC-security of a protocol in the presence of subversion, we formalise the natural properties of a reverse firewall. This allows us to equip existing protocols with reverse firewalls and simplifies the security analysis significantly. We prove that one only needs to show that the firewalls guarantee *correctness*, *strong transparency*, and an *anti-signalling* notion to obtain a subversion-resilient protocol. Hence, we do not need to reprove the UC-security of the underlying protocol and can use existing, efficient protocols. Informally, we prove the following theorem.

  Main Theorem (informal). *Given a protocol base Π', we add a strongly transparent and anti-signalling reverse firewall to it to obtain protocol Π. Then, Π UC-emulates the protocol Π'. Therefore, the protocol Π is subversion-resilient and party-wise secure under the corruptions of Π'.*

- We present a simple UC-secure subversion-resilient OT. In contrast to the only previous subversion-resilient OT of Chakraborty et al. [17], our OT is much simpler, more efficient and also works in the plain UC model instead of the non-plain UC model of [18].
- Additionally, we also show how to use our model to secure commitments against subversion attacks using an easy-to-follow guide. For both functionalities, this leads to quite straightforward security analyses.
- We precisely describe the attack possibilities of a corrupted firewall as impersonating the party. This allows us to give a more fine-granular security analysis avoiding the behaviour used in existing approaches, where the combined party was always treated as maliciously corrupted.
- We present several approaches to circumvent the impersonation attack vector of a corrupted firewall. Some of these are tailored to the specific protocol and thus highly efficient as adding unique signatures to the commitment scheme. Others are more expensive, but also general, as adding zero-knowledge proofs to, e.g., the OT scheme.

1.2 Technical Overview

In this section, we give a technical overview of the results of this paper: Our model for subversion in plain UC; Our model for reverse firewalls in plain UC; How to handle malicious firewalls; How to prove security in the presence of subversion; Subversion-resilient commitments and oblivious transfer. For an illustrative example for readers not familiar with subversion attacks, we will use a running example to explain the used concepts.

Modelling Subversion (Section 2) Our first contribution is to model subversion attacks as *subversion corruptions* in the plain UC model, allowing to reuse a wide range of established results such as the composition theorem. Informally, to subvert a party $\mathcal{P}$ that runs some code π, the attacker $\mathcal{A}$, called subverter, replaces π by a *subverted code* $\bar{\pi}$ by issuing a corresponding subversion command that allows for temporal access to $\mathcal{P}$. After the replacement, this access is disabled and the party strictly follows the (new) code during the remaining protocol run.

Throughout this section, we will use the running example of subverting a UC-secure commitment protocol $\mathtt{Com}$.

In general, the party will employ some software implementation of $\mathtt{Com}$. A subverted implementation $\bar{\mathtt{Com}}$ will now be provided by the subverter. After the implementation was obtained by the party, it will run $\bar{\mathtt{Com}}(v, r)$ on its input v and a uniformly sampled randomness r and output the result c.

A party running $\bar{\pi}$ might behave differently than when running π. However, as the goal of the attacker is to avoid detection, this deviation needs to be limited. For example, since detecting this changed behaviour can lead to an abortion of the protocol by the honest parties or other countermeasures, it (and thus the subversion) should not be noticeable to the honest parties. Hence, we assume the attacker to use an *undetectable subversion* such that the behaviour of $\mathcal{P}$ running π is indistinguishable from the behaviour of $\mathcal{P}$ running $\bar{\pi}$ for other honest parties.

Usually, a subverter is described as wanting to break specific security guarantees of the underlying protocol, but we aim for a more general setting. We thus assume that $\mathcal{P}$ contains some *secret* $\mathtt{sec}$ that the subverter wants to obtain. While this can be, for example, cryptographic key material used to break the security guarantees, it can also be other sensitive information which was stored by other protocols on the device used by the party. To formalise this notion, we say that the subverted code $\bar{\pi}$ is *signalling* when the subverted code can send (or *signal*) some information to the attacker. In other words, $\mathcal{A}$ must be able to distinguish between the behaviour of $\mathcal{P}$ running π and $\mathcal{P}$ running $\bar{\pi}$.

Undetectable Subversions (Section 2.1) While the above informal definition of an undetectable subversion captures the intuitive idea behind subversion attacks nicely, it leaves open what kind of behaviour is detectable and what kind of behaviour is indistinguishable from a run of the non-subverted code.

In general, there are two easy and natural ways to leak sensitive information via subverted implementations. In the first kind of attack, the attacker restricts the attack to only replace the randomness used by the scheme.

One of the strongest attacks uses *rejection sampling*. In such an attack, the subverted implementation samples randomness r_1, r_2, ... until $c_i = \bar{\mathtt{Com}}(v, r_i)$ reveals some important information about sec, e.g., $\mathcal{PRF}_k(c_i) = \mathsf{sec}[0]$ for some PRF-key k known only to the attacker.

The advantage of this kind of attack is that the attacks often work in a somewhat black-box manner, as the subverted code will not care about the other inputs to the functionality. We note that nearly all known subversion attacks do fit into this category and only replace the randomness, e.g., [6, 20, 7, 5, 4, 30, 22, 3, 1, 37, 31, 38, 40, 21].

In the second kind of attack, attackers could also change the actual inputs of the functionality, e.g., by replacing one input by the secret.

To leak information about the secret sec, the attacker could simply commit to sec, i.e., produce a commitment $c = \mathtt{Com}(\mathsf{sec}, r)$ for a truly random r. Now, whenever the commitment c is opened, the receiver (as well as an eavesdropping attacker) will learn the secret sec.

Now, whether such an attack is undetectable highly depends on the situation. For some protocols, the secret might be a perfectly valid input or valid but uncommon enough to raise suspicion or it might even be invalid and thus not maintain the underlying functionality. For example, as discussed by Dodis et al. [24], such behaviour would usually be very obvious in message transmission protocols.

If the value v is usually a completely random value (such as those used in the GMW compiler [26]), the subverted code could produce a commitment $c = \mathtt{Com}(\mathsf{sec} \oplus k, r)$, where k is a key known only to the attacker. However, if v is a highly structured value (such as those in the MPC-in-the-Head zero knowledge proofs [28]), one would require a more advanced technique to embed the secret sec in such values.

Chakraborty et al. [18] decided to treat the second kind of attack behaviour as being undetectable. To counter these attacks, which they denote as *specious*, they require a way to also rerandomise the inputs (even in the ideal world). However, allowing such modifications, they focus on functionalities supporting random inputs. In contrast, we follow (and extend) the approach of Dodis et al. [24] and treat such a behaviour as detectable. We only allow the subverted code to modify the *randomness*, i.e., the attacker can influence the output distribution but the

subverted code must at all times produce valid output. Note that this still gives a wide range of possibilities to the adversary, as they might, e.g., replace random nonces by pseudorandom ones, influence the output distribution of the protocols, or decrease the success probabilities of the protocol. Our attack scenario thus does not capture the case where the replacement of an input would go unnoticed (which has not been used in any real-word subversion [24]). However, we also allow for attackers to, e.g., fix the randomness to a certain value, which might be very obvious and is thus not captured by the model of Chakraborty et al. [18]. Thus, our attacker models are incomparable. Furthermore, our design allows us to work with well-established classic ideal functionalities instead of providing an explicit *subversion interface*.

Protection Mechanisms (Section 3) In order to protect against subversion attacks, several different countermeasures have been proposed in the literature. The two most studied ones are *cryptographic reverse firewalls*, introduced by Mironov and Stephens-Davidowitz [32], and *watchdogs*, introduced by Bellare, Paterson, and Rogaway [5] (although the name was only coined later by Russell et al. [34]). Our second contribution is to model reverse firewalls in plain UC, which aim to actively remove the covert channel established by the subverted implementation leaking the secret. Intuitively, such a firewall is a device used by a cautious user who wants to protect their sensitive secret from exposure due to a subversion attack. The user thus installs the device close to their computer (e.g., by incorporating it into the router) such that the traffic observable by the attacker is available only *after* the data sent from the computer was passed to the firewall. The goal of the cryptographic reverse firewall is to remove the covert channel from the output of the user's computer. Typically, this is done via the means of rerandomisation.

Suppose that the above commitment scheme $\mathtt{Com}$ is additively homomorphic, i.e., $\mathtt{Com}(v,r) \oplus \mathtt{Com}(v',r') = \mathtt{Com}(v \oplus v', r \oplus r')$. If the user sends a commitment via its subverted implementation $c = \tilde{\mathtt{Com}}(v,r)$, this commitment is first given to the firewall before it becomes visible to the attacker. The firewall now samples a random string r' and computes $c' = c \oplus \mathtt{Com}(0,r')$. As r' is sampled uniformly, c' cannot leak sensitive information via the used randomness.

In our model, such firewalls exist as separate parties in the real world. In UC-terminology, such firewalls are modelled as sub-parties of a main party, where the firewalls and main parties each have their own inputs and outputs from other protocol parties. Note that the adversary (the environment) cannot communicate directly with the firewall.

This is a contrast to the model of Chakraborty et al. [18], where the firewall is a direct part of the party to be protected. As we show that subversion attacks are preventable via firewalls in the real world, we do not need to model subversion in the ideal world, hence, these firewalls do not need to exist in the ideal world. This allows us to consider standard ideal functionalities in our setting, similar to the approach of Dodis et al. [24]. Again, this is contrary to the model of Chakraborty

et al. [18], where the ideal functionalities need to have an explicit interface for the firewalls, called the *sanitisation interface*.

The firewalls have a number of important natural properties such as *functionality maintenance*—the firewall should not break the functionality of the protocol run by unsubverted parties; *security preservation*—a subverted party should not be able to break the security guarantees of the protocol; and *exfiltration resistance*—a subverted party should not be able to leak any information related to the secret. A stronger version of the latter property is called *anti-signalling* which, intuitively, guarantees that even the attacker providing the subverted code cannot distinguish whether the party running behind the firewall uses the original code or the subverted code. Another very important property concerns the *transparency* of the firewall, i.e., whether the parties need to be aware of the existence of the firewall. Depending on the party, this transparency might be quite different. For a party $\mathcal{P}$ using a firewall RF, we want a form of *inner transparency* that guarantees that the party does not need to be aware of the existence of (possibly multiple) firewalls. A weaker form of this was introduced by Mironov and Stephens-Davidowitz as *stackability*, guaranteeing that a party could have arbitrarily many firewalls. Similar to the inner transparency, a party $\mathcal{P}'$ interacting with party $\mathcal{P}$ that has firewall RF installed should not need to be aware of the existence of RF. We call this *outer transparency*. Due to the existence of the sanitisation interface in the model of Chakraborty et al., $\mathcal{P}$ needs to be aware of RF and inner transparency is thus not achievable in their model.

Proving Security (Section 4) After we established our models for subversion and cryptographic firewalls, we also develop subversion-resilient protocols. To do so, we assume that we are given a classic protocol Π', which does not use cryptographic firewalls and is thus (potentially) vulnerable to subversion attacks. This protocol securely UC-realises some ideal functionality $\mathcal{F}$. To protect Π' against subversion attacks, we now equip every party $\mathcal{P}_i$ with a firewall RF_i. This gives us a protocol Π. Now, to prove that Π UC-realises $\mathcal{F}$ under subversion-corruption via randomness replacement, we can show that it is sufficient to prove that all firewalls are transparent and anti-signalling and that they keep the correctness of Π'. This implies that Π securely UC-realises Π' and the transitivity of UC-emulation [11] thus implies that Π also UC-realises $\mathcal{F}$. Note that we ignore subversion corruptions in Π' and $\mathcal{F}$, but still provide subversion resilience. Hence, to prove security in our model, we can directly take and adapt existing secure protocols and only need to consider the properties of transparency, anti-signalling, and correctness. In a sense, we are able to transfer the proof technique of Mironov and Stephens-Davidowitz [32] to the UC setting that allows us to treat a transparent firewall as "external". In contrast, Chakraborty et al. [18] needed to "reprove" the security guarantees of Π' when arguing about the security of Π as they bring subversion attacks to the ideal world.

Commitments and OT (full version [2] and Section 5) To show the flexibility of our approach, we instantiate two example functionalities, commitment schemes and oblivious transfer. We consider a commitment scheme due to Canetti, Sarkar, and Wang [13], which is also the basis of the subversion-resilient commitment

scheme of Chakraborty et al. [18]. As explained above, we only need to show correctness, transparency, and anti-signalling and do not need to re-establish the UC-security of the protocol. Hence, our complete security proof for the commitment scheme is only a single page long. We also consider an oblivious transfer protocol due to Canetti, Sarkar, and Wang [13]. The question of the existence of such a concretely efficient subversion-resilient oblivious transfer protocol was raised by Chakraborty, Magri, Nielsen, and Venturi [18] and by Chakraborty, Ganesh, and Sarkar [16]. In the case of the non-plain UC model, this problem was independently solved by Chakraborty et al. [17]. We show that our model allows to present a much simpler and more efficient OT protocol in plain UC.

Malicious Firewalls (full version [2]) A firewall introduces a new attack vector as the firewall itself might be corrupted. While previous works then also treated the combined party as corrupted, we show how to handle this situation in a more fine-grained way and still give (partial) security guarantees.

1.3 Related Work

The concept of reverse firewalls was first introduced by Mironov and Stephens-Davidowitz [32]. They applied it to different primitives in the semi-honest setting against subversions that could also alter the functionality of the underlying primitive. Building upon this, Dodis et al. [24] considered protocols also in the malicious setting, but only against subversions that were functionality-maintaining (due to some strong impossibility results). The primitives secured by firewalls in these works and follow-up works range from oblivious transfer [32], garbled circuits [32], generic constructions against passive attackers [32], CPA-secure encryption [24], key agreement [24], CCA-secure encryption [24], interactive proof systems [25], secure channels [8], and actively secure MPC [14, 15] to oblivious transfer extensions [16]. Li et al. [29] were the first to consider reverse firewalls in a UC-context, but deviate by allowing for an "alarm" signal. The model of Chakraborty et al. [18] does not allow for such an "alarm" and is thus somewhat closer to the original definition of Mironov and Stephens-Davidowitz. A comparison of our model with the model of Chakraborty et al. [18] can be found in Appendix A. Concurrently and independently from this work, Chakraborty et al. [17] presented a subversion-resilient oblivious transfer and password-authenticated key exchange in the model of Chakraborty et al. [18].

While cryptographic reverse firewalls have many advantages, other approaches to prevent subversion attacks exist. For these as well as for a list of the cryptographic primitives used throughout this paper, refer to the full version [2].

2 Subversion Attacks and Countermeasures in UC

The main objective of our paper is to study the subversion of a party. A *subversion* corruption allows an adversary—which is hence called a *subverter* —to change the code π to a *subverted* one $\bar{\pi}$ of a then *subverted*, i.e., subversion corrupted, party.

Unlike other corruption behaviours like malicious or semi-honest corruptions, the adversary has no further access to the party apart from this code replacement. Other information of the party such as a specific secret sec is, hence, not inferable from the subverter directly, but might be extractable through the subverted code. Again, this secret may or may not contain information which, when learned by the adversary, could break the functionality of the protocols. An example secret could be a seed allowing to derive some used key.

2.1 Modelling Subversion Corruptions in UC

In the framework of Universal Composability [11] (for more information, refer to the full version [2]), we model a protocol run via the interaction of several parties $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$. We assume that each party $\mathcal{P}_i$ has some additional sensitive information called sec_i. Each $\mathcal{P}_i$ runs part of the implemented protocol Π in its body, namely π_i. An adversary $\mathcal{A}$ has the goal of gaining information about the secrets of the parties. Therefore, it chooses a set of parties $C \subseteq \{1, \ldots, n\}$ that it wants to *subvert*. To do so, it replaces the code π_i of each party $\mathcal{P}_i$ with $i \in C$ by a subverted version $\bar{\pi}_i$. Remark that the code π_i, or $\bar{\pi}_i$ respectively, has access to sec_i (which might be the empty string if no secret is used). Then, $\mathcal{A}$ chooses another set of parties C' with $C \cap C' = \emptyset$ to corrupt maliciously or semi-honestly. Now, the parties interact following protocol Π and the adversary tries to gain information about all secrets sec_i with $i \in C$.

In the modelling of UC, the environment $\mathcal{Z}$ plays the role of all probabilistic polynomial-time (PPT) adversaries. It can leverage different corruption parameters discussed below which allow to insert further messages, receive internal state messages, and listen to the communication channel depending on the given communication setting. Despite all that, the environment should not be able to distinguish two protocols where one is the idealised goal and the other describes reality as closely as possible.

To model subversions in the framework of UC, the corruption interfaces have to be expanded to handle this new type of corruption. Chakraborty et al. [18] have already presented an adaptation of the UC model that allows modelling subversion attacks and their protections. The authors resorted to only considering undetectable, so-called *specious* subversions and only quantify over specious environments and adversaries. In contrast, we quantify over all PPT environments and introduce a corruption model extending the existing corruption models in the *plain* UC framework by Canetti [11]. This allows reusing well-known important results such as the composition theorem [12] without the need to prove them anew as was necessary for [18]. An additional advantage of this model is that because the model is only extended but not restricted in any way, one can analyse subversion corruption conjointly with other corruption settings.

To analyse a protocol, we give a formal model of the real world protocol which includes both the possible attacks in the form of subversion corruptions by the environment and protective reverse firewalls for each main party. How the firewalls are used as a countermeasure against subversion attacks is discussed in Section 3. However, this newly inserted sub-party will also open up a new

attack vector during the protocol run. As, in reality, a reverse firewall is an additional hardware module, it itself can be manipulated by design or contain an implementation error. To formalise some kind of malicious behaviour, we, hence, give the environment the option to corrupt the firewall. Informally, the subversion corruption of a party Alice with its firewall RF_A is modelled as depicted in Fig. 1, which also shows the insertion of a secret sec by the environment $\mathcal{Z}$.

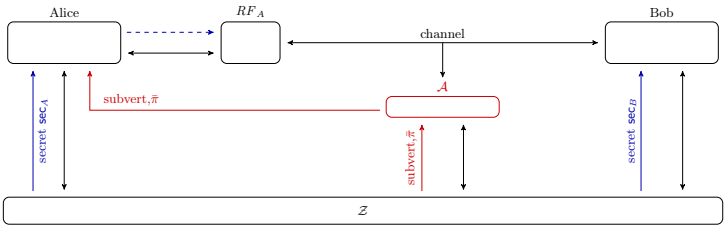


Fig. 1: Informal visualisation of a subversion corruption in the real world in the UC model. Information about the secret might be leaked to the firewall (represented by $\dashrightarrow$). The goal of the firewall is to output messages to the channel that no longer contain this leakage. The leakage of Bob is omitted.

Secret To provide an easier understanding of subversion as a general attack, we claim that every party has a secret sec which we model explicitly. A secret could be some key material from a higher-level protocol, a public-key infrastructure certificate, or the long-term secret family recipe of grandmother's famous apple pie. To model the subversion attack goal, every main party $\mathcal{P}$ is initialised with an empty secret tape, part of its internal state, similar to the randomness tape. More information about modelling the secret is given in the full version [2].

Undetectable Subversion While, in theory, an adversary could arbitrarily subvert a code, some of these changes are very obvious and would raise suspicion. An adversary is thus naturally inclined to only subvert in a way that most communication looks inconspicuous, but capturing this type of subversion is not that simple. Chakraborty et al. [18] tackle this by introducing the definition of *specious* subversions, meaning that no PPT test can distinguish between black-box access to a party running the honest protocol and a party running the subverted protocol except with negligible probability. Their test is not given in a UC definition and, while it helps in understanding this kind of subversion attacks, it is not directly transferable to a corruption behaviour in UC. This essentially results in them only allowing "correct" subversions as well as a sparse amount of trigger attacks. Note that this notion of correctness also encompasses the adversary changing the inputs of the functionality, e.g., by replacing one input by the secret sec. Depending on the protocol, using the secret sec as an

input might be perfectly valid. It could, however, also raise suspicion in case it is a valid but uncommon or even an invalid input, the latter of which would not even maintain the underlying functionality. By allowing to change the inputs and declaring such an attack as being specious, Chakraborty et al. [18] need a way to also rerandomise the inputs (even in the ideal world) and either restrict to functionalities that use random inputs or need to trust the firewall to not change the input (a combination of this can be seen in their GMW compiler).

In contrast, we approach the problem of describing undetectable subversions from a different point of view. Instead of allowing subversions based on how the output distribution of the subverted code looks like, one can restrict subversions based on the changes that are done by the adversary. We choose to focus on a natural way of leaking sensitive information via subverted implementations where the randomness used by the scheme can be changed. By limiting the attack to only this, the output of the subverted implementation is guaranteed to always be correct and, hence, a mere check for correctness of the implementation is not sufficient to detect such a subversion. To the best of our knowledge, there is no paper not satisfying this assumption and there is a variety of subversion attacks that do, e.g., [6, 20, 7, 5, 4, 30, 22, 3, 1, 37, 31, 38, 40, 35]. This also includes all known real-world compromises that were found (see, e.g., Dodis et al. [24, Section 1.1] for a detailed list of such compromises).

Note that only allowing the adversary to change the randomness means that we do not allow changing the inputs of the functionality and such a behaviour is treated as *detectable*. Hence, we do not need to bring the subversion attack vector to the ideal world. We will show that with the help of a firewall, one can already prevent subversion corruptions in the real world by showing indistinguishability with respect to classic real-world protocols and, hence, to well-known ideal UC functionalities. Unlike the approach of Chakraborty et al. [18], we can, therefore, continue to work with the well-established classic ideal functionalities instead of providing an explicit *subversion interface*. However, there are also subversions that we treat as *undetectable* that [18] do not allow. For example, the adversary could change the randomness such that the same constant is used throughout multiple protocol runs. This is clearly not specious, as a test could distinguish such a code from an honest one. Therefore, while the two approaches fall in the category of correct subversions, they are not directly comparable.

As we only consider attacks that change the randomness, we can model them in the plain UC framework as follows: The shell of a party only accepts subversion corruption messages with codes $\bar{\pi}$ which replace at most the randomness function with an adversarially-chosen function. This function has read-only access to the randomness tape and the whole state of the party including its secret `sec`, meaning that it can even take the `sec` as in- and output. The randomness function will then be inserted by the shell into the body such that each call to this so-called `getRandomness`-function is replaced by a call to this `getRandomness`-function of $\bar{\pi}$. However, no other change of the code run by the body is allowed. The function is inserted in the body such that every read access to the randomness tape is replaced by a read access to this adversarially-chosen randomness function.

Note that the shell can easily verify that indeed only the randomness function was replaced. Therefore, we say that $\bar{\pi}$ is only valid if the shell of a subverted party can perform such a check of the validity.

Definition 1 (Subversion via randomness replacement). *Let π be the original code and $\bar{\pi}$ the* subverted code*, where the* `getRandomness`*-function is replaced. Let Π (Π', resp.) be the protocol which runs with the code π ($\bar{\pi}$, resp.). A subversion is detectable if there exists a PPT environment $\mathcal{Z}$ for which $EXEC_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r)$ and $EXEC_{\Pi',\mathcal{A},\mathcal{Z}}(\kappa, r)$ are distinguishable, where $\mathcal{A}$ is the dummy adversary, κ the security parameter, and r the uniformly random chosen randomness.*

From now on, we will refer to the subversion via randomness replacement simply by *subversion*. Whenever we will talk about other types of subversions, we will explicitly say so.

2.2 Preventing Subversion Attacks in UC Using Reverse Firewalls

After formalising the subversion corruption model and the undetectable subversions considered in this work, we now focus on how to protect protocols against such attacks. While different techniques exist in the literature, we focus on the seminal standard of *reverse firewalls* introduced by Mironov and Stephens-Davidowitz [32].

Introduction to Reverse Firewalls Informally, a reverse firewall $RF_{\mathcal{P}}$ (in short only called firewall or RF) is a device additionally installed by a party $\mathcal{P}$ to *protect* the output of their (possibly subverted) implementation from leaking information about the secret, i.e., *signalling*. The firewall obtains all outgoing communication from $\mathcal{P}$ and all ingoing communication to it. Note that while they are two separate identities, we will call $\mathcal{P}$ a (main) party of a protocol Π and $RF_{\mathcal{P}}$ the sub-party of $\mathcal{P}$. If the main party is clear from context, we will simply write RF. A firewall RF is modelled as a sub-party of main party $\mathcal{P}$ in the real world as visualised in Fig. 1. Every main party might have one (or more) firewalls but is not required to have one. In this work, we will analyse the security of party $\mathcal{P}$ gained by its RF in the case of subversion corruption. Therefore, we always have the composed party of $\mathcal{P}$ and RF in mind which we denote by $RF \circ \mathcal{P}$ where all communication is routed via RF. Hence, firewalls communicate via an immediate channel with the respective main parties and, for the communication onward to the network channel, pass on the message after modifying part of the content of the message without modifying the header. For more information about the immediate channels, refer to the full version [2]. The usual setting, which we also use, is the idea that the firewall is somewhat *close* to the party deploying it in such a way that the communication between the party and their firewall can not be eavesdropped.

Subversion Corruption Handling in the Real World As mentioned before, we handle subversion corruptions in such a way that they are already addressed in the real world and not in the ideal world. However, while firewalls allow us to protect parties against subversion attacks, they also bring along possible vulnerabilities, as they can be corrupted by an adversary as well.

Our goal consists in precisely analysing what kind of security guarantee is preserved if even the firewall of a party is corrupted in a certain way. Therefore, we give all environments the additional power of specifying in the corruption parameter not only how the main party is corrupted but also how a potential sub-party should be corrupted. That means that the environment/adversary sends a corruption message with $(\texttt{corrupt}, sid, cp)$ where the corruption parameter cp includes information of the main party and its potentially existing sub-parties, e.g., $cp = (\texttt{subversion} : \bar{\pi}; RF\ \texttt{semi-honest})$. We assume that the shell of a party that does not have a firewall simply ignores the part of cp that does not concern its party. This corruption separation allows us to analyse which kind of security is achievable when, e.g., the firewalls are corrupted. The partial corruption, i.e., corrupting sub-parties of a differently corrupted main party, in the classic UC framework can lead to difficulties as was shown in [27]. However, we want to especially analyse remaining security guarantees while corrupting a main and a sub-party differently. Analysing realistic partial attacks in UC is, e.g., also researched by [9]. For a general understanding of corruption modelling in the UC-setting including the handling via the Corruption Aggregation ITI, we refer to the full version [2] and [11, Sec 7.1].

In this work, we consider corruption parameters $cp = (cp(\mathcal{P}); RF\ \ cp(RF))$ with all kinds of combinations of $cp(\mathcal{P}) \in \{\texttt{honest}, \texttt{malicious}, \texttt{subversion} : \bar{\pi}\}$ and $cp(RF) \in \{\texttt{honest}, \texttt{semi-honest}, \texttt{malicious}\}$. Note that $cp = \texttt{honest}$ means that this party is not corrupted. The different possible corruption parameter combinations of main protocol parties and their firewalls will be translated into a composed classic corruption behaviour. This will then be the behaviour of the corruption parameter for the corresponding main party in a known base protocol which does not have a firewall as sub-party. An overview of the corruption behaviour is given in Table 1 for the corruption transition of a main party $\mathcal{P}$ and its sub-party RF in a protocol Π to its composed classic corruption behaviour. We achieve the classic corruptions of honest, semi-honest and malicious corruptions.

The validity of the results of this table is analysed in Section 4. For this work, the most relevant cases are the ones where the main party $\mathcal{P}$ is subversion corrupted. Interestingly, in the case of a subversion corrupted party with its firewall being only semi-honest, we can show that we achieve a semi-honest composed classic corruption behaviour whereas [18] consider this to be malicious behaviour. An additional interesting result of this work shows that when adapting the protocol, one can hope to achieve an honest composed classic corruption behaviour even if the main party is honest but the firewall behaves maliciously. More details on this are provided in the full version [2].

A subversion corruption allows an adversary to replace the code run by the body of a party. The environment $\mathcal{Z}$ sends a subversion corruption mes-

Table 1: Different corruption combinations and their composed classic corruption behaviour. The lines in grey differ from the table given in [18].

Party	**Firewall**	**Composed Classic Behaviour**
honest	honest / semi-honest	honest
honest	malicious	malicious/honest, see full version [2]
malicious	honest/semi-honest/malicious	malicious
subversion	honest	honest
subversion	semi-honest	semi-honest
subversion	malicious	malicious

sage `subvert` including a code $\bar{\pi}$ to a party $\mathcal{P}$. Upon receiving $\texttt{subvert} = (\texttt{corrupt}, sid, (\texttt{subversion} : \bar{\pi}; RF\ cp(RF))$, the shell of party $\mathcal{P}$ checks if $\bar{\pi}$ is a valid subversion code, i.e., only the `getRandomness`-function is replaced, else the message is ignored. The shell then replaces the randomness function in the body by the adversarially-chosen `getRandomness`-function. Further corruption messages are ignored by the shell of the party. Apart from replacing the code in the body, a subversion corrupted party behaves as an honest party. Note that the provided code might be subverted but does not have to be. Hence, even if $\bar{\pi} = \pi$, we will talk about a subversion taking place. The detailed description of the party's shell code is visualised in the full version [2]. Prior to the randomness function replacement, the shell has to inform the Corruption Aggregation ITI about the corresponding corruption type. To determine the appropriate corruption parameter, the shell then parses the corruption parameter string cp to get the corruption parameter for the main party $\mathcal{P}$ and its firewall RF and follows the transition table shown in Table 1 to get the composed corruption parameter $cp(RF \circ \mathcal{P})$ which it sends to the Corruption Aggregation ITI. By this, the Corruption Aggregation ITI (and therefore the environment $\mathcal{Z}$) will be informed about the composed corruption behaviour in all cases. As one can see, the firewall is a self-contained identity which can be corrupted in a certain manner but as it is a sub-party, it will always behave in a composed setting with the main party. After the activation returns from the Corruption Aggregation ITI, the shell continues according to the corruption parameter $cp(\mathcal{P})$. When the body sends a message to the communication channel, i.e., $\mathcal{F}_{\text{AUTH}}$, the shell directs the message to its RF with the respective corruption parameter $cp(RF)$. The shell of the firewall behaves according to the corruption parameter without informing the Corruption Aggregation ITI. The detailed description of firewalls shell code is also visualised in the full version [2].

Subversion Corruption Handling in the Ideal World To emphasise that subversion attacks are attacks against implementations, our goal is to only consider them in the real world. Therefore, given a standard protocol, we want to prove that its subversion resistance can already be analysed in the real world. We will show

in Section 4 that a real world protocol with subversion corruption and a protection layer consisting of reverse firewalls is indistinguishable from a base protocol that is already proven to be secure in the non-subversion setting. A visualisation of the different protocols and the steps to show the indistinguishability is visible in Fig. 2. By already getting rid of the subversion attack in the real world, we enforce that subversion corruptions are ignored in the ideal world as they are not necessary there. As the adapted real world protocol (a) in Fig. 2 should be indistinguishable from the base protocol (b) in Fig. 2, the environment $\mathcal{Z}$ must not know if a RF exists. Therefore, $\mathcal{Z}$ does not corrupt RF directly and the corruption of RF is not reported to the Corruption Aggregation ITI.

As mentioned in Section 2.1, we assume that every party has a secret, which is modelled by giving all main parties an empty initialised tape where the environment can insert a secret $\mathsf{sec}_{\mathcal{P}}$ via a specific interface. Such an adaptation of the model is not only necessary in the adapted real world protocol but also in every standard real world protocol as well as every ideal functionality. However, note that this is only a modelling artefact due to the corruption handling: The secret must be inserted such that we can UC-simulate the case of a subverted party and a firewall that is semi-honestly or maliciously corrupted. In this case, the firewall would simply leak the secret to the environment and, hence, we model the secret explicitly throughout all protocols.

One could ask how one can see that the ideal functionality still preserves subversion resilience. Subversion corruptions are simply ignored by the ideal functionality such that we can still make use of well-established ideal functionalities. These are then indistinguishable from the standard real world protocol and in turn from the adapted real world protocol which allows subversion corruptions by the environment and protects against them via a reverse firewall.

Remark 1. In this paper, we focus on two-party computation. This allows an easier explanation of our model as, here, the interaction between one party using a firewall and "the outside" can be represented by only communicating with one partner. However, the model is also extendable to multi-party computation as we only look at the plain UC model here. See the full version [2]for more details about the considered corruption.

3 Properties of Reverse Firewalls

In the previous section, we introduced the corruption model and the notion of reverse firewalls, but have not yet described how to actually use such firewalls to protect implementations. In this section, we will formalise the properties of reverse firewalls that are needed to guarantee security. Due to the immense strength of general subversion attacks, it is fairly easy to see that certain subversion attacks are *completely undetectable* and thus avoid all detection mechanisms [6]. Hence, the main idea of the countermeasure studied here is to forgo detection and concentrate on actively removing the covert channel that the subverted implementation aims to open, thereby preventing the leakage of secret information.

3.1 Fundamental Properties of Reverse Firewalls

The firewall itself does not have any secrets and should not gain information about any secret of its main party. Therefore, it can speak of the correctness of a message no more than an eavesdropper. However, the firewall can easily do a syntactical check and only let through messages that have the correct syntax. For example, it can ignore messages consisting of two group elements, if the protocol only expects one group element to be sent.

In [32, 24], the authors not only introduced firewalls as a concept, but also defined different properties that are desirable for such a firewall to have. Informally, we require our firewalls to have the following properties:
Functionality Maintenance: If the implementation is not subverted, the honest firewall should not break the functionality of the underlying protocol.
Security Preservation: If a subverted party is used with an honest firewall, all security guarantees of the underlying protocol should be preserved.
Exfiltration Resistance: The subverted implementation should not be able to leak sensitive information through an honest firewall. This is captured by comparing a subverted party together with an honest firewall against an honest party together with an honest firewall.
The authors of [32, 24] further divide the last two definitions into a strong and a weak version depending on whether this holds against *any* PPT adversary or only against all PPT adversaries that maintain functionality. Such "functionality-maintaining adversaries" correspond to the subversions considered by us. In this paper, we require the firewalls to only fulfil the so-called *weak* properties.

In [32, 24], their firewalls also guarantee the following additional properties:
Stackability: A party should be allowed to have arbitrarily many firewalls. A single correct firewall should already guarantee security.
Transparency: The other parties do not need to be aware of the existence of the firewall(s). This is captured by comparing an honest party together with an honest firewall against an honest party without firewall.

In this work, we will call their [32, 24] notion of transparency *outer transparency*, as it only concerns the other parties, i.e., the view from the outside. Furthermore, we will strengthen the notion of stackability to the notion of *inner transparency*, where we require that an honest implementation needs to behave the same whether a firewall is present or not. We will show that a firewall having both outer and inner transparency (which we call *strong transparency*) allows to keep (nearly) all security guarantees of the underlying protocol.

3.2 Modelling Properties of Reverse Firewalls in UC

While the *functionality maintaining*, *security preserving*, and *exfiltration resisting* properties explained above are clearly needed for reverse firewalls to work at all, there are other useful properties worth to consider. In the following, we will formally define the notions of *transparency* and *anti-signalling* that capture (or extend) the original properties of Mironov and Stephens-Davidowitz [32] in the UC-setting. Furthermore, as we will see, these properties also allow to

simplify the security analysis significantly, as we do not need to "reprove" the UC-simulatability. In the following, Π denotes the run of an incorruptible[6] party $\hat{\mathcal{P}}$ and Π' is the *same* protocol, but run by the composed party $\hat{RF} \circ \hat{\mathcal{P}}$.

Transparency As mentioned above, we extend the definition of *transparency* by [32] and coin the terms *strong transparency* and *outer transparency*. Roughly speaking, the latter is given if an honest party without a firewall is indistinguishable from an honest party together with the firewall behaving honestly, while strong transparency is given if, additionally, the party itself does not have to change its behaviour when using such a firewall.

Definition 2. *(Transparency of Firewalls)* Let $\mathcal{P}$ be a party, RF its firewall, and $\hat{\mathcal{P}}$ (resp. $\hat{RF}$) be the incorruptible versions of $\mathcal{P}$ (resp. RF). Furthermore, let Π be the protocol run by $\hat{\mathcal{P}}$ and Π' be the protocol run by the composed party $\hat{RF} \circ \hat{\mathcal{P}}$. The firewall $\hat{RF}$ provides *outer transparency* if for all PPT environments $\mathcal{Z}$ it holds that $|\Pr[\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r) = 1] - \Pr[\text{EXEC}_{\Pi',\mathcal{A},\mathcal{Z}}(\kappa, r) = 1]|$ is negligible, where $\mathcal{A}$ is the dummy adversary. If, additionally, the protocols Π and Π' contain the same code π for their respective party $\hat{\mathcal{P}}$, then RF provides *strong transparency*.

Strong transparency means that the behaviour does not change regardless of whether $\hat{\mathcal{P}}$ or $\hat{RF} \circ \hat{\mathcal{P}}$ is used. This includes that there is no additional communication to/from the party, so a party does not need to "know" how many firewalls are used or provide different interfaces for different firewalls. Note that this also extends to the behaviour of the firewall, as the behaviour of a composed party $\hat{RF} \circ \hat{\mathcal{P}}$ should not change when employing it together with an additional firewall as $\hat{RF'} \circ (\hat{RF} \circ \hat{\mathcal{P}})$. This directly implies functionality maintenance.

Note that we will only use strong transparency in this work and the notion of outer transparency is mainly introduced to highlight the difference to previous works. Outer transparency can allow for a feedback channel between the firewall and the party, which can weaken the goal of the initial property of strong transparency. For further discussion, see Appendix A.

Non-Signalling Composed Party Another property to model is the notion that the firewall counteracts the leakage of a subverted party by making the combination of party and firewall *non-signalling*. The goal is that not even the adversary can distinguish the behaviour of $RF \circ \bar{\mathcal{P}}$ from $RF \circ \mathcal{P}$. We call such a firewall *anti-signalling* (similar to *exfiltration-resistance* by Mironov and Stephens-Davidowitz [32] and *strong sanitation* by Chakraborty et al. [18]).

Definition 3. *(Anti-Signalling Firewalls)* Let $\mathcal{P}$ be a party and RF its firewall. Let $\bar{\mathcal{P}}$ be the subvertable version of $\mathcal{P}$. Let $\hat{\mathcal{P}}$ and $\hat{RF}$ the incorruptible version of $\mathcal{P}$ and RF, respectively. Let Π be the protocol run by the composed party

[6] I.e., a party ignoring the corruption messages send by the adversary.

$\hat{RF} \circ \bar{\mathcal{P}}$. Let Π' be the protocol run by $\hat{\mathcal{P}}$ and $\hat{RF}$. The firewall RF is anti-signalling if for all PPT environments $\mathcal{Z}$ it holds that $|\Pr[\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r) = 1] - \Pr[\text{EXEC}_{\Pi',\mathcal{A},\mathcal{Z}}(\kappa, r) = 1]|$ is negligible, where $\mathcal{A}$ is the dummy adversary.

In most applications, there are *perfectly* anti-signalling firewalls, i.e., the runs $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r)$ and $\text{EXEC}_{\Pi',\mathcal{A},\mathcal{Z}}(\kappa, r)$ are distributed identically.

Now, if we are able to construct an anti-signalling firewall providing strong transparency, the interaction of a subverted party using an honest firewall with another party is indistinguishable from the interaction of honest parties. The notion of anti-signalling allows to replace the subverted party using the honest firewall with an honest party using the honest firewall and the strong transparency allows to replace the honest party using the honest firewall by an honest party.

4 Security Analysis of Subversion Corrupted Protocols with Reverse Firewalls

In this section, we want to show how one analyses the security of a protocol susceptible to subversion corruptions that is at the same time protected by firewalls. Therefore, we will utilise specific properties of firewalls discussed in Section 3 to give a generic security proof. We will focus on the adaptation of existing UC-secure protocols, where, similar to all other previous approaches, a combined party containing a malicious firewall will be treated as malicious. In the full version [2], we show another approach, where we need to adapt the protocols, but can also guarantee security against corrupted firewalls.

Subversion corruptions are an attack vector to real-world protocols. However, we can already get rid of this attack vector in a hybrid step following the corruption behaviour of the transition table in Table 1 in the real world, such that we do not need to forward subversion corruptions to the ideal world. This gives us the advantage that we can easily make use of given standard UC functionalities without reproving elementary theorems of plain UC. Examples for basic relevant protocols, namely oblivious transfer, will be provided in the next section.

4.1 How to Analyse Subversion Resilience of a Protocol

In general, we want that a protocol $\Pi \geq_{\text{UC}} \mathcal{F}$ under subversion corruption. Therefore, our main focus lies in proving that a protocol Π already UC-emulates a classic protocol Π' under subversion corruption in the real world. By showing that subversion corruptions are *not* necessary in the ideal world and keeping the transitivity of UC-emulation in mind, we get that Π', and thereby Π, UC-realises an ideal functionality $\mathcal{F}$ under subversion corruption. For a more in-depth explanation, we go through the following steps:

First. We assume that there exists a classic protocol Π' ((b) in Fig. 2) which will be the base protocol of Π ((a) in Fig. 2), meaning that no subversion corruptions (and no firewalls) are considered. In particular, Π' UC-realises the desired ideal functionality $\mathcal{F}$ ((c) in Fig. 2) under (semi-honest and) malicious corruption.

Second. To begin with, we give every main party an interface to receive a secret sec from the environment. Note that in a real computer system, the secret could also already be in the state of the party. By giving $\mathcal{Z}$ the power to set the secret, we model the adversary to be even more powerful. For the security analysis, the `secret` interface has to be added to every protocol, i.e., even to the base protocol and the ideal functionality. This has no influence on the security behaviour apart from the modelling aspect of subversion resilience. Not only does this adaptation explicitly model the subversion goal of leaking a secret, it also is necessary to be able to simulate the case of a corrupted firewall, which in turn is vital to analyse the advantages and disadvantages of such an additional protection layer.

Third. We adapt Π' such that we can protect it against subversion attacks. All adaptations follow the model of Section 2, i.e., subversion corruptions and sub-party corruptions of the firewall are possible. This allows us to analyse its security under subversion corruption. Additionally, following the modelling of Section 2.2, we give a firewall RF as a sub-party to every main party $\mathcal{P}$ which could leak something about its secret via subversion during the protocol run. The firewall should be strongly transparent and anti-signalling, a combination of properties which, as we will show, allows to protect against subversion corruptions.

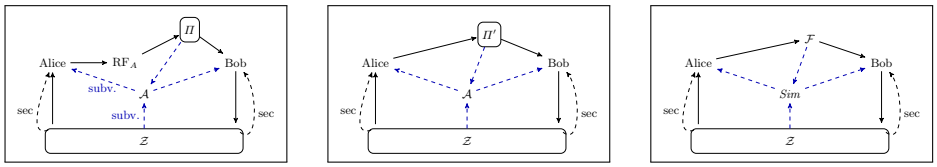
These protocol adaptations yield the protocol Π ((a) in Fig. 2).

Fourth. We show that the adapted protocol Π and the base protocol Π' are indistinguishable, i.e., we give a simulator $Sim_{\Pi'}$ such that for all PPT environments $\mathcal{Z}$ it holds that $\Pr[\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} = 1] - \Pr[\text{EXEC}_{\Pi',Sim_{\Pi'},\mathcal{Z}} = 1]$ is negligible. We show that $\Pi \geq_{\text{UC}} \Pi'$ under party-wise subversion, semi-honest, and malicious corruption following the corruption behaviour transition given in Table 1.

Fifth. From the transitivity of UC-emulation [11, Sec. 4.3] it follows that if $\Pi \geq_{\text{UC}} \Pi'$ and $\Pi' \geq_{\text{UC}} \mathcal{F}$, then $\Pi \geq_{\text{UC}} \mathcal{F}$. More discussion regarding the transitivity of UC-emulation of our notion is given in the full version [2].

Formally speaking, we want to prove our following main theorem, which states that the adapted protocol Π is indistinguishable from the base protocol Π'. This indicates that both protocols are resilient against subversion corruptions via the help of reverse firewalls. The proof can be found in the full version [2].

Theorem 1 (Main Theorem). *Let Π be the protocol with main parties $\mathcal{P}_i$ under party-wise subversion, semi-honest, and malicious corruption, and the*



(a) Adapted Protocol (b) Base Protocol (c) Ideal World

Fig. 2: The adapted protocol (a) of the base protocol (b) models subversion corruption and firewalls. Protocol (b) UC-realises the ideal functionality in (c).

(sub-) parties $RF_{\mathcal{P}_i}$ under party-wise semi-honest and malicious corruption in the $\{\mathcal{F}_{CRS}, \mathcal{F}_{AUTH}\}$-hybrid model. Let Π' be the protocol with main parties $\mathcal{P}'_i$ under party-wise semi-honest and malicious corruption in the $\{\mathcal{F}_{CRS}, \mathcal{F}_{AUTH}\}$-hybrid model. Let the corruption type of the parties $\mathcal{P}_i$ and $RF_{\mathcal{P}_i}$ transition from their individual party-wise corruption to their composed classic corruption behaviour type following the corruption transition Table 1. Let $RF_{\mathcal{P}_i}$ be strongly transparent and anti-signalling. Let r be the randomness and κ be the security parameter. There exists a simulator $Sim_{\Pi'}$ such that for all PPT environments $\mathcal{Z}$, we have $EXEC_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r) \approx EXEC_{\Pi',Sim_{\Pi'},\mathcal{Z}}(\kappa, r)$, where $\mathcal{A}$ is the dummy adversary.

Proof Idea of Theorem 1. When proving Theorem 1, we need to show that $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r) \approx \text{EXEC}_{\Pi',Sim_{\Pi'},\mathcal{Z}}(\kappa, r)$, where $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r)$ is the execution of the adapted protocol ((a) in Fig. 2) and $\text{EXEC}_{\Pi',Sim_{\Pi'},\mathcal{Z}}(\kappa, r)$ is the execution of the base protocol ((b) in Fig. 2). In the following, we give the proof idea for the most interesting cases. For the case of an honest main party $\mathcal{P}$ and an honest firewall RF in Π, we can show that this is indistinguishable from an honest party $\mathcal{P}$ in Π' if RF fulfils the notion of strong transparency (Definition 2). The case where the RF is semi-honest works similarly as an honest $\mathcal{P}$ does not leak any secret. Hence, it translates to an honest $\mathcal{P}$ in Π'. For the case of a subverted main party $\mathcal{P}$ and an honest firewall RF in Π, we can show that this is indistinguishable from an honest party $\mathcal{P}$ in Π' if RF fulfils the notion of anti-signalling (Definition 3). The firewall RF cannot offer protection in the case of a maliciously corrupted main party $\mathcal{P}$, thus, this directly translates to the maliciously corrupted case without RF. For the case of a malicious firewall RF in Π, it can impersonate the honest or subverted main party $\mathcal{P}$. Thus, we need to treat the composed party as malicious. For countermeasures against this, we refer to the full version [2] for a discussion of how one can translate an honest $\mathcal{P}$ with malicious RF in Π to an honest party in Π'. The case of a subverted $\mathcal{P}$ and a semi-honest RF in Π is indistinguishable from a semi-honest $\mathcal{P}$ in Π' mainly because in Π', the simulator learns the secret of the semi-honestly corrupted party as it is part of the party's state.

A more extensive description of each of these cases is given in the full version [2], where we provide an argument for every line in Table 1, i.e., all possible cases given the static corruption combinations. For the most interesting cases of subversion corrupted parties, we will provide a simulator $Sim_{\Pi'}$ in the full version, which also provides insights into the communication flow. There, we also give further discussions of the corruption handling, including the steps required to get to the ideal world. For completeness, there, we also provide the code for the shell of a party in the base protocol Π'. Note that its main difference to a general shell of party $\mathcal{P}$ in a standard protocol is the interface to insert a secret. With the shell code behaviour of $\mathcal{P}$ and RF in the adapted protocol Π, one can see the indistinguishability regarding the corruption handling via the Corruption Aggregation ITI. Additionally, one can see that the transitivity of UC-emulation from the plain UC framework is directly applicable. Further details on the transitivity of UC-emulation are discussed in the full version. Finally, we explain the steps to achieve subversion-resilience in our model in the full version.

Corollary 1. *Given Theorem 1, it follows that* $\Pi \geq_{UC} \mathcal{F}$.

From the transitivity of UC-emulation [11] and Theorem 1, it follows directly that $\Pi \geq_{\mathrm{UC}} \mathcal{F}$ because $\Pi \geq_{\mathrm{UC}} \Pi'$ and $\Pi' \geq_{\mathrm{UC}} \mathcal{F}$.

5 Oblivious Transfer

To show how to adapt protocols to be subversion resilient, we consider the example of an oblivious transfer (OT) protocol. A similarly simple protocol for commitments can be found in the full version [2]. The protocols are expanded to include firewalls designed to rerandomise all randomness used by the parties. The security can then be shown in two steps: it has to be proven that the firewall is strongly transparent and anti-signalling and, additionally, the new protocol should still be correct. We focus on the case where an honest party with a malicious firewall is treated as malicious, which is consistent with the handling by Chakraborty et al. [18]. Yet, in our model, the capabilities of such an impersonated party are strictly weaker than those of a completely malicious party. Notably, as long as the party is honest, the combined party cannot leak the secret. In the full version, we give a detailed analysis of this impersonation scenario covering multiple solutions, depending on the protocol and its security goals.

Oblivious transfer is one of the open problems stated by [18] and [16]. Independently from this work, such a subversion-resilient OT was provided by Chakraborty et al. [17] in the aforementioned subversion model of [18]. While much more involved in the model of Chakraborty et al., we show that a relatively simple and efficient OT protocol already suffices for subversion-resilience in our plain UC model.

The sender S who knows two messages m_0, m_1 communicates with the receiver $\mathcal{R}$ who has a choice bit b and wants to learn m_b. The goal of the interaction is to allow the receiver to learn this message, and this message only, without the sender learning anything about b. The OT protocol consists of two rounds with each party sending one message to each other. As both of the messages are generated using randomness, this requires two different firewalls.

We base our protocol on the UC-secure 1-out-of-2 OT protocol given by [13] where one of the messages $m_0, m_1 \in \mathbb{G}$ from the sender S can be obtained by the receiver $\mathcal{R}$. The protocol has to be adapted by making the field element $T_2 \in \mathbb{G}$ part of its CRS. The group $\mathbb{G}$ with generator g and the field $\mathbb{Z}_q$ are public inputs. While security under augmented semi-honest corruption (i.e., semi-honest corruption where the adversary can choose the input of the corrupted party) is given, plain semi-honest security of the base OT scheme in [13] has not been shown yet. The real run of the oblivious transfer scheme is illustrated in Fig. 3.

For a more formal description of the protocol, a detailed analysis including the ideal functionality and the proof for Theorem 2, refer to the full version [2].

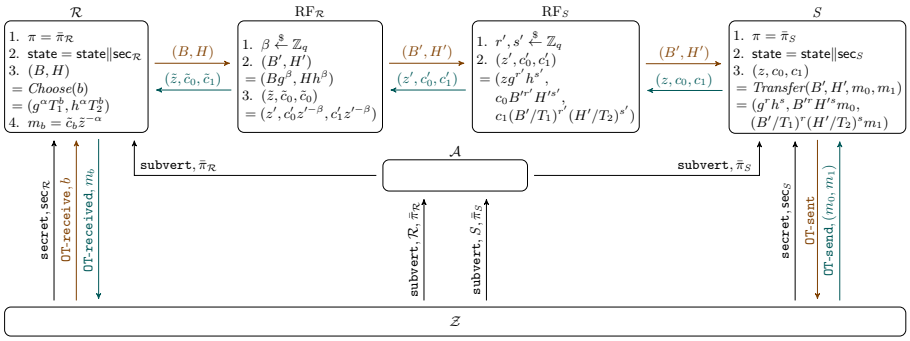


Fig. 3: Real run of the oblivious transfer scheme (omitting $\mathcal{F}_{\text{CRS}}$ and $\mathcal{F}_{\text{AUTH}}$). Here, $Choose(b)$ is used to embed the choice bit b, while $Transfer(B', H', m_0, m_1)$ then embeds the values m_0, m_1 to be transferred.

Theorem 2. *The protocol Π_{srOT} UC-realises the ideal functionality $\mathcal{F}_{OT}$ in the $\{\mathcal{F}_{CRS}, \mathcal{F}_{AUTH}\}$-hybrid model, where the main parties are maliciously, semi-honestly or subversion corrupted with a subversion code $\bar{\pi}$, and the respective sub-parties RF are honest, semi-honest or malicious.*

6 Conclusion

In this work, we provide a new model to analyse the security of given protocols against subversion attacks within the plain Universal Composability (UC) framework and show how to integrate reverse firewalls as a countermeasure. More specifically, our model effectively captures realistic subversion scenarios where only the randomness of a party can be subverted. Then, given a real world *base* protocol securely UC-realising an ideal functionality, we prove that our resulting subversion-resilient protocol UC-realises this ideal functionality as well. Thereby, our model allows to argue subversion-resilience of already existing UC-secure protocols without necessitating additional security proofs. Our results rely on new definitions of properties for reverse firewalls, mainly strong transparency and anti-signalling, enabling robust security guarantees even when these firewalls are compromised. To demonstrate the practicality of our model, we applied it to commitment schemes and oblivious transfer protocols.

Based on the contributions of this paper, several directions for future research remain open: A natural extension would be to explore the applicability of the model to other cryptographic protocols. In particular, ensuring subversion-resilience in post-quantum cryptographic settings would strengthen the future security landscape. A more detailed analysis of impersonation attacks—or other possibly opened attack vectors—by the reverse firewalls as well as extending the framework to handle adaptive adversaries who can dynamically corrupt and subvert different components of a system could also enhance its robustness.

Acknowledgments. We thank the anonymous reviewers from Asiacrypt 2024, PKC 2025, and ACNS 2025 for their valuable feedback which yield to this improved work. We thank Johannes Ottenhues for proofreading and for his helpful discussions and suggestions.
This work was supported by funding by the German Federal Ministry of Education and Research (BMBF) under the project "Sec4IoMT" (ID 16KIS1692 and 16KIS1693), and by KASTEL Security Research Labs.

References

1. Armour, M., Poettering, B.: Algorithm substitution attacks against receivers. Int. J. Inf. Sec. **21**(5), 1027–1050 (2022)
2. Arnold, P., Berndt, S., Müller-Quade, J., Ottenhues, A.: Protection Against Subversion Corruptions via Reverse Firewalls in the Plain Universal Composability Framework, Cryptology ePrint Archive, Paper 2023/1951 (2023).
3. Ateniese, G., Magri, B., Venturi, D.: Subversion-Resilient Signature Schemes. In: CCS, pp. 364–375. ACM (2015)
4. Bellare, M., Jaeger, J., Kane, D.: Mass-surveillance without the State: Strongly Undetectable Algorithm-Substitution Attacks. In: CCS, pp. 1431–1440. ACM (2015)
5. Bellare, M., Paterson, K.G., Rogaway, P.: Security of Symmetric Encryption against Mass Surveillance. In: CRYPTO (1). Lecture Notes in Computer Science, pp. 1–19. Springer (2014)
6. Berndt, S., Liskiewicz, M.: Algorithm Substitution Attacks from a Steganographic Perspective. In: CCS, pp. 1649–1660. ACM (2017)
7. Berndt, S., Wichelmann, J., Pott, C., Traving, T., Eisenbarth, T.: ASAP: Algorithm Substitution Attacks on Cryptographic Protocols. In: AsiaCCS, pp. 712–726. ACM (2022)
8. Bossuat, A., Bultel, X., Fouque, P., Onete, C., van der Merwe, T.: Designing Reverse Firewalls for the Real World. In: ESORICS (1). Lecture Notes in Computer Science, pp. 193–213. Springer (2020)
9. Broadnax, B., Koch, A., Mechler, J., Müller, T., Müller-Quade, J., Nagel, M.: Fortified Universal Composability: Taking Advantage of Simple Secure Hardware Modules. IACR Cryptol. ePrint Arch. (2018)
10. Canetti, R.: Universally Composable Security. J. ACM **67**(5), 28:1–28:94 (2020)
11. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. IACR Cryptol. ePrint Arch. (2000)
12. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: FOCS, pp. 136–145. IEEE Computer Society (2001)
13. Canetti, R., Sarkar, P., Wang, X.: Efficient and Round-Optimal Oblivious Transfer and Commitment with Adaptive Security. In: ASIACRYPT (3). Lecture Notes in Computer Science, pp. 277–308. Springer (2020)
14. Chakraborty, S., Dziembowski, S., Nielsen, J.B.: Reverse Firewalls for Actively Secure MPCs. In: CRYPTO (2). Lecture Notes in Computer Science, pp. 732–762. Springer (2020)
15. Chakraborty, S., Ganesh, C., Pancholi, M., Sarkar, P.: Reverse Firewalls for Adaptively Secure MPC Without Setup. In: ASIACRYPT (2). Lecture Notes in Computer Science, pp. 335–364. Springer (2021)

16. Chakraborty, S., Ganesh, C., Sarkar, P.: Reverse Firewalls for Oblivious Transfer Extension and Applications to Zero-Knowledge. In: EUROCRYPT (1). Lecture Notes in Computer Science, pp. 239–270. Springer (2023)
17. Chakraborty, S., Magliocco, L., Magri, B., Venturi, D.: Key Exchange in the Post-snowden Era: Universally Composable Subversion-Resilient PAKE. In: ASIACRYPT (5). Lecture Notes in Computer Science, pp. 101–133. Springer (2024)
18. Chakraborty, S., Magri, B., Nielsen, J.B., Venturi, D.: Universally Composable Subversion-Resilient Cryptography. In: EUROCRYPT (1). Lecture Notes in Computer Science, pp. 272–302. Springer (2022)
19. Checkoway, S., Niederhagen, R., Everspaugh, A., Green, M., Lange, T., Ristenpart, T., Bernstein, D.J., Maskiewicz, J., Shacham, H., Fredrikson, M.: On the Practical Exploitability of Dual EC in TLS Implementations. In: USENIX Security Symposium, pp. 319–335. USENIX Association (2014)
20. Chen, R., Huang, X., Yung, M.: Subvert KEM to Break DEM: Practical Algorithm-Substitution Attacks on Public-Key Encryption. In: ASIACRYPT (2). Lecture Notes in Computer Science, pp. 98–128. Springer (2020)
21. Cheng, J., Wang, Y., Chen, R., Huang, X.: Subverting Cryptographic Protocols from a Fine-Grained Perspective- A Case Study on 2-Party ECDSA. In: ACISP (2). Lecture Notes in Computer Science, pp. 370–390. Springer (2024)
22. Degabriele, J.P., Farshim, P., Poettering, B.: A More Cautious Approach to Security Against Mass Surveillance. In: FSE. Lecture Notes in Computer Science, pp. 579–598. Springer (2015)
23. Discussion about Kyber's tweaked FO transform, (2023). https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/WFRDl8DqYQ4. Discussion Thread on the PQC mailing list.
24. Dodis, Y., Mironov, I., Stephens-Davidowitz, N.: Message Transmission with Reverse Firewalls - Secure Communication on Corrupted Machines. In: CRYPTO (1). Lecture Notes in Computer Science, pp. 341–372. Springer (2016)
25. Ganesh, C., Magri, B., Venturi, D.: Cryptographic reverse firewalls for interactive proof systems. Theor. Comput. Sci. **855**, 104–132 (2021)
26. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In: STOC, pp. 218–229. ACM (1987)
27. Hofheinz, D., Shoup, V.: GNUC: A New Universal Composability Framework. J. Cryptol. **28**(3), 423–508 (2015)
28. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-Knowledge Proofs from Secure Multiparty Computation. SIAM J. Comput. **39**(3), 1121–1152 (2009)
29. Li, G., Liu, J., Zhang, Z., Zhang, Y.: UC-Secure Cryptographic Reverse Firewall-Guarding Corrupted Systems with the Minimum Trusted Module. In: Inscrypt. Lecture Notes in Computer Science, pp. 85–110. Springer (2021)
30. Lin, Y., Chen, R., Wang, Y., Wang, B., Liu, L.: Substitution Attacks Against Sigma Protocols. In: CSS. Lecture Notes in Computer Science, pp. 192–208. Springer (2022)
31. Marchiori, D., Giron, A.A., do Nascimento, J.P.A., Custódio, R.: Timing analysis of algorithm substitution attacks in a post-quantum TLS protocol. In: Anais do XXI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, pp. 127–140 (2021)
32. Mironov, I., Stephens-Davidowitz, N.: Cryptographic Reverse Firewalls. In: EUROCRYPT (2). Lecture Notes in Computer Science, pp. 657–686. Springer (2015)

33. Perlroth, N, Larson, J, Shane, S: Secret documents reveal NSA campaign against encryption. New York Times. September **5** (2013)
34. Russell, A., Tang, Q., Yung, M., Zhou, H.: Cliptography: Clipping the Power of Kleptographic Attacks. In: ASIACRYPT (2). Lecture Notes in Computer Science, pp. 34–64 (2016)
35. Russell, A., Tang, Q., Zhu, J.: Crooked Indifferentiability of the Feistel Construction. In: ASIACRYPT (6). Lecture Notes in Computer Science, pp. 427–460. Springer (2024)
36. Simmons, G.J.: The history of subliminal channels. IEEE J. Sel. Areas Commun. **16**(4), 452–462 (1998)
37. Teseleanu, G.: Unifying Kleptographic Attacks. In: NordSec. Lecture Notes in Computer Science, pp. 73–87. Springer (2018)
38. Young, A.L., Yung, M.: Kleptography: Using Cryptography Against Cryptography. In: EUROCRYPT. Lecture Notes in Computer Science, pp. 62–74. Springer (1997)
39. Young, A.L., Yung, M.: The Dark Side of "Black-Box" Cryptography, or: Should We Trust Capstone? In: CRYPTO. Lecture Notes in Computer Science, pp. 89–103. Springer (1996)
40. Young, A.L., Yung, M.: The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems. In: CRYPTO. Lecture Notes in Computer Science, pp. 264–276. Springer (1997)

A Comparison to Chakraborty et al.

In this section, we will give a more detailed comparison of our work and model to the model introduced by Chakraborty, Magri, Nielsen, and Venturi [18] and also used by Chakraborty, Magliocco, Magri, and Venturi [17].

The main difference in our view of subversion attacks can be summed up in the question of "which attacks are undetectable?".

Let us reconsider the two subversion attacks against commitments described in Section 1: In the first attack, *rejection sampling* was used to find good randomness such that $c = \bar{\mathtt{Com}}(v, r)$ reveals some information about the secret to the subverter. In the second attack, the subverted code changed the input of the commitment and compute $c = \bar{\mathtt{Com}}(\mathsf{sec}, r)$ instead.

In contrast to the model of Chakraborty et al. [18], we consider the second attack, i.e., subversions that change an input of the underlying algorithm, to be detectable. The reason for this is that the party running the subverted implementation might notice that the value in the commitment is not equal to the value that was supposed to be there by inspecting the output of the commitment. This kind of inspection closely resembles the *watchdogs*, introduced by Bellare, Paterson, and Rogaway [5]. Hence, the party might become suspicious and thus detect the presence of the subverted code.

This difference thus result in the major difference between our models. In our model, subversion attacks do not change the functionality of the underlying protocol and are simply implementation attacks. This allows our ideal world to have neither subverted implementations nor firewalls and lets us realise standard known functionalities that guarantee security without asking for firewalls. That the ideal world does not model any firewalls is possible as the environment cannot

communicate with RF in the real world either. Hence, in our model, where only the randomness can be subverted, the analysed commitment functionality $\mathcal{F}_{\texttt{Com}}$ used is simply the standard functionality, where the committing party sends a value v to commit to the functionality. Later on, it can also open this value to another party.

In contrast, in the model of Chakraborty et al. [18], the firewall does also exist in the ideal world and ideal functionalities thus need to have an explicit interface for the firewalls, called the *sanitisation interface*. For example, the commitment functionality $\mathcal{F}_{\texttt{Com}}$ of Chakraborty et al. [18] thus works in three steps. First, the committing party sends value v to commit to the functionality. Then, the reverse firewall is informed that some values was sent and can send a blinding string r via the sanitisation interface of the functionality. The functionality now changes the stored value from v to $v \oplus r$ and informs the committing party about this by sending r. Finally, when the commitment is opened, the updated value $v \oplus r$ is revealed to the receiving party.

A.1 Transparency

In Section 3.2, we introduced the notions of *outer transparency* and *strong transparency*. Suppose that the parties $\mathcal{P}$ and $\mathcal{P}'$ communicate via some protocol Π where $\mathcal{P}$ might use a firewall. Informally, outer transparency means that $\mathcal{P}'$ cannot decide whether the party $\mathcal{P}$ uses the firewall or not. Strong transparency also requires something one might call *inner transparency*, where $\mathcal{P}$ itself does not need to be aware of the existence of the firewall.

Outer transparency implies that based on the messages alone, other parties cannot infer whether a firewall is used. Note that outer transparency requires the firewall to not change the functionality. However, this is not true for the other way around as a functionality-maintaining firewall could, for example, add a bit as a prefix to every message which would not break functionality but outer transparency. In [18], the firewalls are modelled to have an additional input through which the content of its party's message can be changed. However, there are also firewalls without this input, for which a definition of *transparency* is given. Informally, such a firewall is transparent if it cannot be distinguished from a firewall that does not modify the communication and directly passes the message along. This indistinguishability is similar to our definition of outer transparency. However, strong transparency does not follow from their definition since, as discussed below, that would disallow the usage of feedback channels.

The authors of [32, 24] used transparency notions similar to ours to argue about the security of protocols in the case of a corrupted firewall: The *security against the firewall* [32] should follow directly from the security of the underlying protocol by considering the power of a corrupted firewall to be the same as the power of a maliciously corrupted communication partner. In our terminology, their idea is the following: Suppose that Π is secure against malicious corruptions. Then, the security of $\mathcal{P}$ can be argued by treating RF together with $\mathcal{P}'$ as a malicious party. Due to the inner transparency, the protocol run for $\mathcal{P}$ is indistinguishable from a run without the firewall against a malicious party $\mathcal{P}'$

and the security of the protocol thus implies the security of $\mathcal{P}$ (even when RF is corrupted). Similarly, the security of $\mathcal{P}'$ can be argued by treating RF together with $\mathcal{P}$ as a malicious party. Due to the outer transparency, the protocol run for $\mathcal{P}'$ is indistinguishable from a run without the firewall against a malicious party $\mathcal{P}$ and the security of the protocol thus implies the security of $\mathcal{P}'$ (even when RF is corrupted).

While the argument above neglects the implications of an impersonation attack, this is still the general idea we keep in mind. In our model that is using a shell and a body for each party, it is the task of each shell to handle the respective in- and outputs to both the body and the respective recipient, whether that is another shell of the same party or a shell of another party, e.g. a firewall or (in the case of no firewall) the communication partner. Hence, the body of a party does not need to be aware of the existence of a firewall and, therefore, does not need to change its content. It is now easy to see why inner transparency, corresponding to the fact that the body behaves identically whether there exists a firewall or not, is achievable in our setting. Consequently, this approach allows us to make similar arguments to guarantee security of the party as [32, 24].

However, in the model of Chakraborty et al., there is a need for a communication feedback channel between the firewall and its party. In the above commitment example, this feedback channel is used to inform the party about the blinding factor r that was used. As the party thus needs to be aware of the existence of the firewall, firewalls cannot provide inner transparency in the model of Chakraborty et al., and thus also not guarantee strong transparency. In this case, the argument does not hold and this feedback channel gives a corrupted firewall another possibility of sending corrupted messages to its party. Whether this additional attack vector allows an adversary to extract sensitive information or not depends on the specific handling of this, a statement as general as when not using a feedback channel is not possible.

A.2 Corrupted Firewalls

A firewall RF is modelled as a sub-party of main party $\mathcal{P}$ in the real world as visualised in Fig. 1. Every main party might have one (or more) firewalls but is not required to have one. Firewalls communicate via an immediate channel with the respective main parties and, for the communication onward to the network channel, pass on the message after modifying part of the content of the message without modifying the header.

For example, in the case of an authenticated channel, Alice would send $(\texttt{send}, sid, Alice, Bob, msg)$ via the immediate channel to its firewall, which adapts the message to msg' and sends $(\texttt{send}, sid, Alice, Bob, msg')$ to the ideal authenticated channel functionality $\mathcal{F}_{\text{AUTH}}$, were msg' is a version of the message m where the possible covert channel is removed (often via rerandomisation).

This gives a good model of how messages are routed in a real-life network that passes messages/packets along a sequence of routers. Here, each device in a network only needs to know the address of its own router. Remark that RF sends messages in the name of $\mathcal{P}$ which could be used for an impersonation attack by

a malicious RF, which is discussed further in the full version [2]. Apart from the interfaces for the input from its main party and the network channel, no further inputs from the outside are modelled/accepted, i.e., the environment $\mathcal{Z}$ cannot give further input to the firewall.

As a RF opens up a new attack vector to the setting, as they might also get corrupted, we provide corruption interfaces. Corruption handling and interfaces are part of the model handling rather than of the communication protocol. Further details are given in the attack Section 2.2. The environment can only make use of the corruption interfaces of RF. Note that RF does not provide a secret interface, as RF should and would not know a secret.

This model additionally differs from the one presented in [18] in that, there, an ideal functionality $\mathcal{F}$ has an explicit interface for the firewall RF, as $\mathcal{F}$ will be adapted to be "sanitisable". This means that $\mathcal{F}$ has a set of interfaces to handle the in- and output from a party and a set of interfaces to handle the changes made by RF. Consequently, a party and its firewall cannot directly talk to each other as they have to communicate through the ideal functionality resulting in the requirement that all parties must use a firewall. On the modelling side, this change requires that every $\mathcal{F}$ must be adapted if it is used in a hybrid model; the authors of [18] propose a wrapper for their functionalities. Here, one can directly see that $\mathcal{F}$ learns the secret of the party as it has transferred the plain message from party to firewall.

The benefit of firewalls is using them to protect already existing protocols against subversion attacks. Hence, it is intuitive to only consider functionality-maintaining firewalls as these do not change the underlying functionality. As already discussed above, in [18], the firewalls are allowed to modify the main party's inputs. This heavily impacts the definition of a functionality. Consider a commitment scheme as an example. Employing such a firewall that might change the committed element would lead to the commitment of a random value. While this is sufficient for functionalities using random inputs, more general functionalities might require that a specific value is committed. One could argue that in this case, a firewall could simply be provided with inputs that do not change the value. However, this would give the firewalls—more specifically the inputs of the firewalls—the power to decide on the obtainable functionality. Hence, one could end up with two protocols for the *exact same* functionality, but with very different correctness guarantees depending on firewalls used.

Note that there is also an ambivalence present regarding informing the party of the value that was actually committed to. As can be seen in [18], a protocol implementing such an ideal functionality can be designed in such a way that no update is passed back to the party. While this means that no feedback channel is needed, the party can also no longer use the committed value later on in the protocol, as it is unknown to the party. If, instead, an update should always be returned to the party to allow this kind of functionality, a feedback channel is necessary. Since this channel is not needed in the underlying protocol, this requires a corresponding modification. Furthermore, using multiple firewalls with a feedback channel together requires additional management: All firewalls have

to forward all received randomnesses to the party which in the end has to sum all of them up. Alternatively, the firewalls have to be adapted to wait for their outer firewall to answer with a randomness before updating their own randomness accordingly and passing it to next firewall such that the party will receive only one updated randomness.