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

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

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

Abstract. While many modern cryptographic primitives have stood the test of time, attacker have already begun to expand their attacks beyond classical cryptanalysis by specifically targeting implementations. One of the most well-documented classes of such attacks are subversion (or substitution) attacks, where the attacker replaces the implementation of the cryptographic primitive in an undetectable way such that the subverted implementation leaks sensitive information of the user during a protocol execution. The revelations of Snowden have shown that this is not only a dangerous theoretical attack, but an attack deployed by intelligence services. Several possible solutions for protection against these attacks are proposed in current literature. Among the most widely studied ones are *cryptographic reverse firewalls* that aim to actively remove the covert channel leaking the secret. While different protocols supporting such firewalls have been proposed, they do no guarantee security in the presence of concurrent runs. This situation was resolved by a recent work of Chakraborty et al. (EUROCRYPT 2022) that presented the first UC-model of such firewalls. Their model allows to provide security if a subverted party uses an honest firewall. 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. Hence, the authors needed to reprove fundamental theorems such as the composition theorem. Finally, the high complexity of their model also makes designing corresponding protocols a challenging task, as one also needs to reprove the security of the underlying protocol.
In this paper, we present a simpler model capturing cryptographic reverse firewalls in the plain UC model. The simplicity of our model allows to also reason about corrupted firewalls and still maintain strong security guarantees. Furthermore, we resolve the open question by Chakraborty et al. (EUROCRYPT 2022) and by Chakraborty et al. (EUROCRYPT 2023) and present the first direct UC-secure oblivious transfer protocol along with a cryptographic reverse firewall.

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 [1]. Such attacks were later studied more formally under the name of *kleptography* by Young and Yung [2, 3]. 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 [4]. Furthermore, later revelations due to Snowden exposed the existence of NSA's *Project Bullrun*. Some goals of these 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*"[5]. These revelations led to an increased alertness by cryptographic researchers that quickly started to develop possible countermeasures against such *subversion attacks* [6, 7]. 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[4] was proposed in the standardisation process of post-quantum cryptography to prevent such attacks [8], but was ultimately rejected.

Unfortunately, there are subversion attacks that are *completely undetectable* in a black-box setting [9] and all countermeasures thus need to use a non-black-box approach. One of the most widely studied countermeasures are the so-called *reverse firewalls* that aim 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. [10] 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

[4] called the *hash of shame*

universal composability (UC) model by Canetti [11, 12] to capture subversion attacks and countermeasures build 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 GMW compiler [13]. 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 [14, 15]. 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. 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. Similar, Chakraborty et al. model a subverted implementation together with a semi-honestly corrupted firewall as a maliciously corrupted combined party. Here, the attacker might actually learn about sensitive information due to the subversion, but should not be able to deviate from the protocol run. Again, the model of Chakraborty et al. needs to over-approximate the abilities of an attacker. Furthermore, due to the complex generalisation of the already complex UC model, the authors need to reprove several fundamental results such as the *composition theorem* and leave more advanced functionalities such as oblivious transfer (OT) as future work. Finally, as the authors use a generalization of the UC model, to convert known UC-secure protocols to their model they need to essentially *reprove* the security of the protocol.

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 capture and minimise the attack vectors available to a corrupted firewall precisely. 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.
– 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 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*, *transparency*, and *anti-signalling* to obtain s subversion-resilient protocol. Hence, we do not need to reprove the UC-security of the underlying protocol and can use existing, efficient protocols.
– We present an easy-to-follow guide on how to use our model. We show how this leads to a quite simple analysis by considering the case study of commitments.
– We present the first direct UC-secure subversion-resilient OT and thus answer the open questions raised by Chakraborty, Magri, Nielsen, and Venturi [10] and by Chakraborty, Ganesh, and Sarkar [16]. Concurrently, and independently from this work, a subversion-resilient OT was also provided by Chakraborty et al. [17], but in the model of [10].
– 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 over-approximations used in existing approaches, where the combined party always was treated as maliciously corrupted.
– We present several approaches to circumvent the impersonate 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. Some are more expensive, but also general, as adding zero-knowledge proofs, e.g., to 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. Along the way, we also compare our model with the model of Chakraborty et al. [10].

Modeling Subversion (Section 3) Our first contribution is to model subversion attacks 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 of the code, this access is disabled making the party strictly follow the (new) code during the remaining run of the protocol.

Throughout this paper, we will use the running example of submitting a commitment scheme.

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

Due to running $\bar{\pi}$, the party might behave differently than when running π. As the goal of the attacker is to avoid detection, this deviation needs to be somewhat 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 detectable by these honest parties. Hence, to avoid detection, we assume the attacker to use a *specious subversion*, i.e., the behaviour of $\mathcal{P}$ running π should be indistinguishable from the behaviour of $\mathcal{P}$ running $\bar{\pi}$ for other honest parties in the protocol.

Usually, a subverter is described as wanting to break some security guarantees of the underlying protocol, but we aim for a more general setting. We thus assume that $\mathcal{P}$ contains some *secret* 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 $\bar{\pi}$.

The goal of the subverter is to obtain information about the secret $\mathtt{sec}$, which might be a cryptographic key used in the protocol or confidential data not directly related to the cryptographic protocol.

Speciousness (Section 3.2) While the above informal definition of *specious* subversion captures the intuitive idea behind subversion attacks nicely, it leaves open what kind of behaviour is suspicious and what kind of behaviour is indistinguishable from a run of the non-subverted code. In general, there are two natural ways to leak sensitive information via subverted implementations. In the first kind of attack, the attacker restricts the attack to only change 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 s, e.g., $\mathcal{PRF}_k(c_i) = s[0]$ 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 here that nearly all known subversion attacks do fit into this category and only change the randomness, e.g., [9, 18, 19, 6, 7, 20, 21, 22, 23, 24, 25, 2, 26].

In contrast to the above attack, attackers could also only change the actual inputs of the functionality, e.g., by replacing one input by the secret $\mathtt{sec}$.

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

Now, whether such an attack is unsuspicious highly depends on the situation. Depending on the protocol, the secret $\texttt{sec}$ might be a perfectly valid input, uncommon enough to raise suspicion, or it might be invalid and thus not maintain the underlying functionality.

If the value v is a completely random value (such as those used in the GMW compiler [13]), the subverted code could produce a commitment $c = \texttt{Com}(\texttt{sec} \oplus k, r)$, 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 [27]), one would require a more advanced technique to embed the secret s in such values.

Chakraborty et al. [10] decided to treat this behaviour of changing the inputs as being non-suspicious and thus declare such an attack as being specious. They thus needed a way to also rerandomise the inputs (even in the ideal world) and allowing such modifications restricts to using functionalities that have random inputs (such as the GMW compiler [13]). In contrast, we treat such an behaviour as suspicious behaviour. To formalise the notion that we treat such attacks as non-specious, we only allow the subverted code to modify the used *randomness*, i.e., a specious code must at all times produce *valid* output. Due to this separation between the inputs and the randomness, we do not need to model subversions in the ideal world. Hence, while this modification *weakens* the attacker model of Chakraborty et al. [10], it allows us to work with the well-established classical ideal functionalities instead of providing an explicit *subversion interface.*

Protection Mechanisms (Section 4) In order to protect against subversion attacks, several different countermeasures have been proposed in the literature. The two most studied are *cryptographic reverse firewalls*, introduced by Mironov and Stephens-Davidowitz [14], and *watchdogs*, introduced by Bellare, Paterson, and Rogaway [6] (although the name was coined later by Russell et al. [28]). Our second contribution is to model reverse firewalls introduced by Mironov and Stephens-Davidowitz [14] in plain UC, which aim to actively remove the covert channel established by the subverted implementation that leaks the secret. Intuitively, such a firewall is a device used by a cautious user that wants to protect their sensitive secret from exposure due to a subversion attack. The user thus installs the devices close to their computer (e.g., by incorporating it into the router) in such a way 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. Most 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 $c = \bar{\mathtt{Com}}(v, r)$, 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, i.e., a party and their firewall each have their own inputs, outputs and their own corruption handling.

The commitment functionality $\mathcal{F}_{\mathtt{Com}}$ used in our work 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.

This is a contrast to the model of Chakraborty et al. [10], where the firewall is a direct part of the party to be protected. As described above, we do not need to model subversion in the ideal world, hence these firewalls do not need to exist in the ideal world which allows us to consider standard ideal functionalities in our setting. This is again, contrary to the model of Chakraborty et al. [10], where the ideal functionalities need to have an explicit interface for the firewalls, called the *sanitization interface*.

The commitment functionality $\mathcal{F}_{\mathtt{Com}}$ of Chakraborty et al. [10] 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 sanitization 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.

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 the secret. A stronger version of the latter property is called *anti-signalling* and, intuitively, guarantees that even the attacker providing the subverted code can not 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*, which guaranteed that a single 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 sanitization interface in the model of Chakraborty et al., the party $\mathcal{P}$ needs to be aware of the firewall and the notion of inner transparency is thus not achievable in their model.

Malicious Firewalls (Section 8) On the first glance, transparency might look like a feature that is clearly nice to have, but not of particularly importance for obtaining security guarantees. However, as we show in this paper, transparent firewalls are very useful to guarantee the security in the case that the party $\mathcal{P}$ is honest, but the firewall RF is (maliciously) corrupted. In the model of Chakraborty et al. [10], the firewall and the party are treated as a single party and hence, this single party needs to be treated as corrupted. Using a more fine-granular analysis, the authors note some security guarantees might be preserved in this setting and thus establish a new corruption model called *isolation*, but treat this as malicious corruption throughout their work. Hence, the installation of a firewall might actually *weaken* the security of the complete system.

In our model, the party and the firewall are two separate parties and we can thus analyse this important scenario in more detail. Suppose that an honest party $\mathcal{P}$ using a corrupted firewall RF interacts with another party $\mathcal{P}'$ that can either be honest or corrupted. We first consider the security guarantees of $\mathcal{P}'$. Due to the outer transparency of the firewall, the party $\mathcal{P}'$ does not need to be aware of the existence of RF. We can thus treat the *combination* of $\mathcal{P}$ and RF as a single corrupted party $RF \circ \mathcal{P}$ and the active security of the underlying protocol thus gives the security guarantees of $\mathcal{P}'$. A similar argument is also used by Chakraborty et al.

Now, consider the security guarantees of $\mathcal{P}$. Due to the inner transparency of the firewall, the party $\mathcal{P}$ does not need to be aware of the existence of RF. We can thus treat the *combination* of RF and $\mathcal{P}'$ as a single corrupted party $\mathcal{P}' \circ RF$ and the active security of the underlying protocol thus gives the security guarantees of $\mathcal{P}$. As described above, this argument does not work in the model of Chakraborty et al., as $\mathcal{P}$ needs to be aware of the existence of RF. Hence, the installation of a firewall can not weaken the security of the complete system in our model.

However, when taking a closer look at the definition of transparency, there is one problem left: While the corrupted firewall can not extract sensitive information from $\mathcal{P}$, it can *impersonate* $\mathcal{P}$ due to the outer transparency. Hence, as RF is allowed to send messages in the name of $\mathcal{P}$, a corrupted firewall can perform impersonation attacks. This problem is somewhat unavoidable due to the requirement of outer transparency and thus also already applies to the original model of cryptographic firewalls due to Mironov and Stephens-Davidowitz [14] as well as Chakraborty et al. [10]. For protocols not relying on authentication, this is not a problem (and only such protocols were studied previously such as commitments or key exchange). However, when considering more general protocols, such behaviour might actually weaken the security guarantees. To handle this issue explicitly, we introduce a new ideal functionality, called $\mathcal{F}_{\text{chAUTH}}$ which explicitly covers the possibilities of impersonation that such a corrupted firewall can have. We also show two realisations of this functionality: A very efficient one for our commitment protocol based on signatures and a more general version using zero-knowledge proofs for our oblivious transfer protocol.

Proving Security (Section 5) After we established our models for subversion and cryptographic firewalls, we also develop subversion-resistant protocols. To do so, we assume that we are given a classical 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 with specious codes, we give a very useful theorem. This theorem allows us to show that it is sufficient to show that all firewalls are transparent and anti-signalling and that they keep the correctness of Π' to obtain this strong subversion-resilience. This implies that Π securely UC-realises Π' and the transitivity of UC-emulation [29] thus implies that Π also UC-realises $\mathcal{F}$.

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 contrast, Chakraborty et al. [10] needed to “reprove” the security guarantees of Π' when arguing about the security of Π.

Commitments and OT (Section 6 and Section 7) To show the flexibility of our approach, we consider two important functionalities, commitment schemes and oblivious transfer. First, we consider a commitment scheme due to Canetti, Sarkar, and Wang [30], which is also the basis of the subversion-resilient commitment scheme of Chakraborty et al. [10]. 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 [30] and, again, only need to consider correctness, transparency, and anti-signalling. Our complete security proof is thus shorter than two pages. The existence of such an concretely efficient subversion-resilient oblivious transfer protocol thus answer open questions raised by Chakraborty et al. [10].

1.3 Related Work

As mentioned above, several protocols and corresponding cryptographic reverse firewalls have been proposed. The primitives secured by such firewalls range from oblivious transfer [14], garbled circuits [14], generic constructions against passive attacker [14], CPA-secure encryption [15], key agreement [15], CCA-secure encryption [15], interactive proof systems [31], secure channels [32], and actively secure MPC [33, 34] to oblivious transfer extensions [16]. A similar approach to reverse firewalls was presented by Alwen, shelat, and Visconti [35]. They make use of a so called *mediator*, who is also allowed to rerandomize messages, to obtain so called *collusion-free* protocols, as introduced by Lepinski, Micali, and shelat [36]. However, in contrast to the reverse firewalls, such a mediator is able to intercept the messages of *all* parties (and not only those that deployed the firewall) and, furthermore, it can actively exchange messages with all parties in arbitrary order. Hence, such a mediator is a much stronger concept than reverse firewalls. We refer the reader to Mironov and Stephens-Davidowitz [14] for a more detailed comparison.

Li et al. [37] were the first to consider reverse firewalls in a UC-context. However, their definition strongly differs from the original definition due to Mironov and Stephens-Davidowitz, as they also incorporate the possibility that the firewall detects a subversion. In this case, the firewall "alarms", which directly halts the complete protocol. In some sense, their model thus uses a mix of reverse firewalls and watchdogs (discussed below). Furthermore, they consider their firewalls to be uncorruptable and make use of random oracles to make the firewalls somewhat deterministic.

The model of Chakraborty et al. [10] does not allow for such an "alarm" and is thus somewhat closer to the original definition of Mironov and Stephens-Davidowitz. The technical overview already compares our model to this model. 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. [10].

While cryptographic reverse firewalls have many advantages, other approaches to prevent subversion attacks exist.

Watchdogs: The notion of a *watchdog* was introduced by Bellare, Paterson, and Rogaway [6]. A watchdog can either be *offline* or *online*. An offline watchdog is given the (possibly subverted) implementation along with an honest reference implementation and before the user uses the (possibly subverted) implementation, the watchdog can perform certain black-box tests to find differences between the two implementations. If it can't find any difference, the user is free to use the implementation. An online watchdog can additionally also observe the complete communication while the implementation is in use in order to detect deviant behaviour. Watchdogs have also been used to secure different primitives such as one-way-permutations [28], pseudorandom generators [28], randomness generators [38, 28], public-key encryption schemes [38, 39], authenticated encryption [40], random oracles [41], and signature schemes [42, 28, 43].

Alternative Countermeasures: Both Fischlin and Mazaheri [44] and Abdolmaleki et al. [45] assume the existence of a non-compromised first phase where the parties can freely exchange information. Fischlin and Mazaheri use this phase to exchange an honestly signed message which then is used to create subversion-resilient signatures. Abdolmaleki et al. use this first secure phase to exchange initial randomness from which they can derive non-subverted randomness for the remaining zero-knowledge protocol. Finally, Badertscher et al. [46] also consider a kind of subversion attacks in a UC-setting, but, in contrast to nearly all of the other mentioned papers, their implementation do not try to be undetectable. They show how to perform updates in such a scenario that return the implementation to a non-subverted state and consider digital signatures and zero-knowledge proofs.

2 Preliminaries

In this section, we described the needed preliminaries and notations used throughout this work.

Notation We denote component-wise multiplication with $\times$, e.g. $(a, b) \times (c, d) = (a \cdot c, b \cdot d)$. The term PPT will be used for Probabilistic Polynomial Time, i.e., efficient randomised algorithms. Computational indistinguishability is denoted by $\approx$.

Glossary Throughout this paper, we use the following notations unless stated otherwise:

$\mathcal{P}$	a party
$\mathcal{P}'$	a second party
RF	a firewall which is itself also a party
$RF \circ \mathcal{P}$	a composed party consisting of a party $\mathcal{P}$ and a firewall RF
$\hat{\mathcal{P}}$	an incorruptible party $\mathcal{P}$
$\bar{\mathcal{P}}$	a subverted party $\mathcal{P}$
π	a code
Π	a protocol

2.1 Cryptographic Primitives

In this work, we will consider multiple cryptographic primitives. Here, we only give an informal overview and will present the needed technical definitions throughout the paper.

Commitment: A *commitment scheme* consists of a single PPT algorithm $\texttt{Com}$ that is given some value $v \in \{0, 1\}^*$ and produces a commitment c. The commitment is called *hiding* if no attacker can distinguish between commitments of v and commitments of another value v', even if v and v' are provided by the attacker. The commitment is called *binding* if it is infeasible to find two random strings r and r' and two different values v and v' such that $\texttt{Com}(v, r) = \texttt{Com}(v', r')$.

Oblivious Transfer: An *oblivious transfer* (OT) consists of two PPT algorithms $(S, \mathcal{R})$. The *sender* S knows two messages m_0 and m_1 and the *receiver* $\mathcal{R}$ knows a bit b. After interaction between S and $\mathcal{R}$, the receiver should obtain the message m_b without learning anything about the other messages m_{1-b}. Furthermore, S should not learn anything about the bit b.

Signature Schemes: A *signature scheme* consists of three PPT algorithms $\mathtt{Kgen}$, $\mathtt{Sign}$, and $\mathtt{Vfy}$. The *key generation algorithm* $\mathtt{Kgen}$ is given a security parameter 1^κ and produces the verification key vk and the signing key sk. The *signing algorithm* $\mathtt{Sign}$ is given the signing key sk and a message $msg \in \{0,1\}^*$ and produces a signature σ. Finally, the *verification algorithm* $\mathtt{Vfy}$ is given the verification key vk, a message msg, and a signature σ and outputs a bit $b \in \{0,1\}$. For all outputs σ of $\mathtt{Sign}(sk, msg)$, we need that $\mathtt{Vfy}(vk, msg, \sigma) = 1$. A signature scheme is called *universal unforgeable* if all attackers with oracle access to $\mathtt{Sign}(sk, \cdot)$ are not able to produce a valid pair (m, σ) (except for those provided by the oracle).

Zero-Knowledge proofs: A *zero-knowledge proof* for a language $\mathcal{L} \subseteq \{0,1\}^*$ consists of a pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$. Both parties know some *instance* $x \in \{0,1\}^*$ and the *prover* $\mathcal{P}$ knows a witness proving that $x \in \mathcal{L}$ and wants to convince the *verifier* $\mathcal{V}$ that $x \in \mathcal{L}$. A pair $(\mathcal{P}, \mathcal{V})$ is called *complete*, if an honest prover can (nearly) always convince a verifier and *sound* if a dishonest prover can (nearly) never convince a verifier. Finally, the system is said to have the *ZK property* if even a dishonest verifier does not learn any information about the witness held by the honest prover.

2.2 Universal Composability

The traditionally employed game-based security notions have the disadvantage that they usually cannot capture *compositions.* To still guarantee security in such quite involved situation, Canetti [11, 12] introduced the *universal composability* (UC) model. To formalise security in the UC model, one defines an *ideal functionality* $\mathcal{F}$ that captures the ideal behaviour of the primitive. To *implement* such an ideal functionality, one formalises a protocol Π and then shows through a simulation that a real world execution of this protocol is indistinguishable from a run of the ideal functionality. In that case we will say the protocol securely UC-realises the ideal functionality. A more detailed description of the model is given below:

We are given n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ which are interactive Turing machines (ITMs) with codes $\pi_1, \ldots, \pi_n$. These parties can be *corrupted* by the attacker $\mathcal{A}$ that sends certain corruption messages to the parties to corrupt them, e.g., maliciously. The corrupted parties are now somehow controlled by $\mathcal{A}$, although their behaviour might be restricted by their corruption type.

To achieve security under general composition, the framework introduces a distinguisher $\mathcal{Z}$, called the *environment* which can interact with the executions by, e.g., providing inputs and interacting with the attacker $\mathcal{A}$. We denote a real-world execution of the protocol Π with the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ using the security parameter κ and the randomness r by $\mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r)$. This execution is compared with a run of the ideal functionality $\mathcal{F}$ where all parties are dummy parties controlled by the environment $\mathcal{Z}$. The goal is now to construct a *simulator Sim* that can interact with the ideal functionality such that the real execution is indistinguishable from the ideal execution $\mathrm{EXEC}_{\mathcal{F},Sim,\mathcal{Z}}(\kappa, r)$.

Formally, this is defined as:

Definition 1. *UC-security.* A protocol Π UC-realises an ideal functionality $\mathcal{F}$ (in symbols $\Pi \geq_{\mathrm{UC}} \mathcal{F}$) if for all PPT attackers $\mathcal{A}$, there exists a PPT simulator *Sim* such that for all PPT environments $\mathcal{Z}$, we have that $\mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r)$ is computationally indistinguishable from $\mathrm{EXEC}_{\mathcal{F},Sim,\mathcal{Z}}(\kappa, r)$ (with respect to the security parameter κ). Here, r is drawn randomly.

Note that if both executions are real world protocols, a simulator showing the indistinguishability lets the one protocol *UC-emulate* the other one.

Furthermore, all parties can also communicate with other ideal functionalities $\mathcal{F}'$. Such a situation is called the $\mathcal{F}'$-hybrid model. The *composition theorem* now allows to replace this ideal functionality $\mathcal{F}'$ by any protocol Π' UC-realising $\mathcal{F}'$ while keeping all of the security guarantees preserved. In the following we will explain some hybrid functionalities that we will utilize.

The ideal functionality $\mathcal{F}_{\mathrm{CRS}}$ models a common reference string that is accessible by all parties.

Definition 2 (The Ideal Functionality $\mathcal{F}_{\mathbf{CRS}}$, adapted from [47]). *$\mathcal{F}_{CRS}$ proceeds as follows, when parametrised with a distribution D.*

1. *When activated for the first time on input $(\mathtt{value}, sid)$ by a party $\mathcal{P}$, choose a value $d \leftarrow D$ and generate a public delayed output $(\mathtt{value}, sid, d)$ for $\mathcal{P}$. Answer subsequent* `value` *inputs from parties $\mathcal{P}'$ by generating a public delayed output $(\mathtt{value}, sid, d)$ for $\mathcal{P}'$.*

The ideal functionality $\mathcal{F}_{\mathrm{AUTH}}$ models an authenticated channel between a sender S and a receiver $\mathcal{R}$.

Definition 3 (The Ideal Functionality $\mathcal{F}_{\mathbf{AUTH}}$, adapted from [48]).

1. *Upon invocation, with input $(\mathtt{send}, sid, S, m)$ from ITI S, send backdoor message $(\mathtt{sent}, sid, S, \mathcal{R}, m)$ to the adversary.*
2. *Upon receiving backdoor message $(\mathtt{OK}, sid)$: If not yet generated output, then output $(\mathtt{sent}, sid, S, \mathcal{R}, m)$ to $\mathcal{R}$.*
3. *Upon receiving backdoor message $(\mathtt{corrupt}, sid, m', \mathcal{R}')$, record being corrupted. Next, if not yet generated output then output $(\mathtt{sent}, sid, S, \mathcal{R}', m')$ to $\mathcal{R}'$.*
4. *On input $(\mathtt{reportcorrupted}, sid)$ from S: If corrupted, output* `yes` *to S, else output* `no`*.*
5. *Ignore all other inputs and backdoor messages.*

3 Modelling Subversion Attacks in UC

After giving an introduction to subversion attacks, we will formally model them in UC. Then, motivated by an adversary that wants such an attack not to be noticed by the subvertee, *specious subversions* will be defined, while showing the damage such an attack can still achieve. Lastly, a summary of the specific type of subversion attacks is provided that we will consider in the rest of the paper.

3.1 Introduction to Subversion Corruption

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. Apart from this code-exchange, the adversary has no access to the party. Other information of the party including a specific sec is, hence, not inferable from the party 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 and, e.g., could be a seed allowing to derive a used key.

In a general setting, we can model this, informally, by the following situation where the subversion of a single party is being depicted in Figure 1: The parties

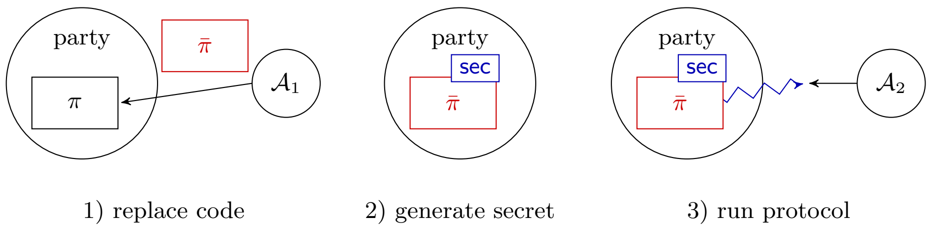


Fig. 1: The adversary can replace the code π of a party with a speciously subverted one $\bar{\pi}$ which can then leak information about the secret to the adversary. Step 1) and 2) might be switched depending on the type of subversion discussed below.

$\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ interact via a set of functionalities (resp. protocols). Each party $\mathcal{P}_i$ has some additional sensitive information called sec_i. Depending on the considered scenario, as discussed below, this secret is generated before or after a corruption has taken place.

Now, each $\mathcal{P}_i$ runs part of the implemented protocol Π in its body, namely π_i. The adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ has the goal of gaining information about the secrets of the parties. Therefore, $\mathcal{A}_1$ 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 if this exists within the party.

The adversary $\mathcal{A}_2$ chooses another set of parties C' with $C \cap C' = \emptyset$ to corrupt maliciously or semi-honestly. Now, the parties interact and the adversary tries to gain information about all secrets sec_i with $i \in C$.

Remark 1 (Comparison subversion and semi-honest corruption.). In a *subversion* corruption, the adversary can choose the behaviour of the party in the beginning with knowledge about neither the input values nor the future states of the computation. This includes the activation status implying that the subverter cannot halt the party from the outside. In contrast, a semi-honest corruption allows learning all these information but gives no power to change the behaviour. In theory, it might thus be possible to combine the two corruption models to achieve semi-honest subversion corruptions. However, this would give an adversary direct access to the secret, making the whole goal of leaking the secret obsolete.

3.2 Subversion Corruption Model in UC

The corruption interfaces are expanded to the handling of subversion corruption as described in Section 3.2.

While Chakraborty et al. [10] have already presented a corruption model extension that allows modelling subversion attacks and its protections in the UC framework, they resorted to only quantifying 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* framework of Universal Composability by Canetti [48]. This allows us to reuse well-known important results such as the composition theorem [49] without the need to prove them anew as was necessary for [10]. 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.

Subversion Corruption Handling in the Real World The real-world handling of a subversion corruption can be seen in Figure 2.

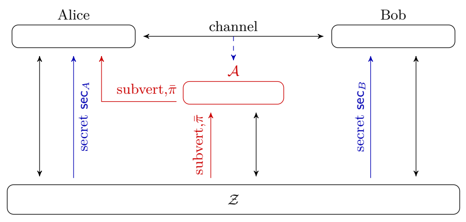


Fig. 2: Sketch of a subversion corruption in the real world. Leaked information about the secrets transmitted over the channel can be learned by the adversary (represented by $\dashrightarrow$).

For a general understanding of the corruption modelling in the UC-setting, we refer to Section 2.2 and Section 7.1.1 in [48]. A subversion corruptions allows an adversary to exchange the code run by the body of a party: The adversary $\mathcal{A}$ sends a subversion corruption message `subvert` combined with a—presumably subverted—code $\bar{\pi}$ to a party $\mathcal{P}$. Upon receiving $(\texttt{corrupt}, sid, \texttt{subversion}, \bar{\pi})$ from $\mathcal{A}$, the shell of party $\mathcal{P}$ checks if $\bar{\pi}$ is a valid subversion code, else the message is ignored. The shell then inserts the validated code at the requested place in the body and sets a "subversion corruption" flag. While in general, the whole code of the body could be replaced, this code must be efficiently checkable of its subversion properties. This will later be explained in more detail. Further corruption messages are ignored by the shell of the party. Apart from exchanging the code in the body, a subversion corrupted party behaves as an honest—not corrupted—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.

To model the attack goal, every party is initialised with an empty secret tape, which is part of the state, similar to the randomness tape. This tape is used to store a secret `sec` that is inserted by the environment via the `insertSecret`-interface at any time during the protocol run. This tape can also be updated again via the secret interface; this entails that the whole state can be moved to the secret.

Static subversion corruption. If we are in the static corruption model, the party sends upon invocation a notification message to the adversary. The subversion corruption message `subvert` has to be received by the party from the adversary in the very next activation. Only then can a secret be generated, hence, the case illustrated in Figure 1 fits the static subversion corruption case.

Extracting the `sec`*.* As the adversary has no access to the subverted party during the protocol run, it needs access to the communication generated by the party. This could either be by eavesdropping on the communication channel, which can then be maximally *authenticated*, or by maliciously corrupting the communication partner of the subverted party.

Subversion Corruption Handling in the Ideal World As subversion attacks are attacks against implementations, they should only exist in the real world. A subversion corruption thus only invokes the following in the ideal world: On receiving the subversion corruption message `subvert` and a code $\bar{\pi}$ for a party $\mathcal{P}$, the simulator informs the ideal functionality that $\mathcal{P}$ is subversion corrupted. Noteworthy, neither the presumably subverted code nor the $\texttt{sec}_{\mathcal{P}}$ is received by the ideal functionality, and the behaviour of all parties, e.g. protocol and backdoor messages are unchanged.

The general handling of a subversion corruption in the ideal world can be seen in Figure 3. When handling semi-honest corruptions, the ideal functionality additionally receives the secrets $\texttt{sec}_A$ and $\texttt{sec}_B$.

Specious Subversion In general, there are two natural ways to leak sensitive information via subverted implementations. In the first type of attack, the

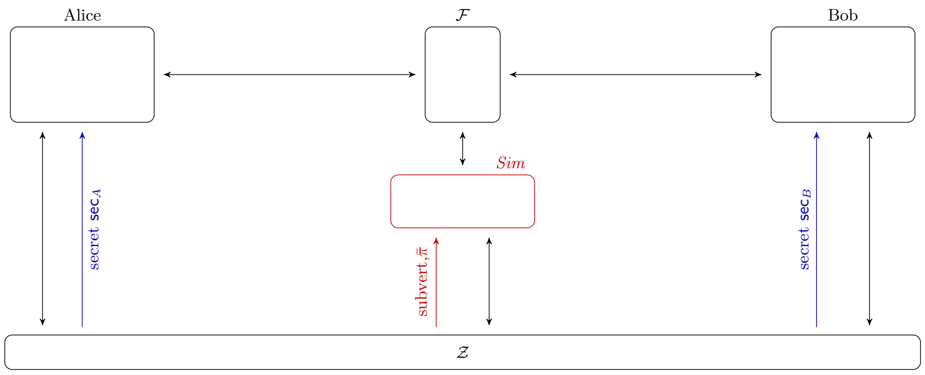


Fig. 3: Sketch of a subversion corruption in the ideal world.

attacker restricts the attack to only change the randomness used by the scheme, which guarantees that the output of the subverted implementation is always *correct*. Hence, to detect such a subversion, a mere check for correctness of the implementation is not sufficient. Nearly all known subversion attacks do fit into this category, e.g., [9, 18, 19, 6, 7, 20, 21, 22, 23, 24, 25, 2, 26].

In the second type of attack, the actual inputs of the functionality could be changed, e.g., by replacing one input by the secret s. Depending on the protocol, this might be highly suspicious, as the secret s might be a perfectly valid input, uncommon enough to raise suspicion, or it might be invalid and thus not maintain the underlying functionality. As noted above, Chakraborty et al. [10] decided to treat this behaviour of changing the inputs as being non-suspicious and thus declare such an attack as being specious. They thus needed a way to also rerandomise the inputs (even in the ideal world) and are restricted to functionalities that use random inputs (such as the GMW compiler [13]). In contrast, we treat such an behaviour as suspicious behaviour. To formalise the notion that we treat such attacks as non-specious, we only allow the subverted code to modify the used *randomness*, i.e., a specious code must at all times produce *valid* output. Due to this separation between the inputs and the randomness, we do not need to model subversions in the ideal world. Hence, while this modification *weakens* the attacker model of Chakraborty et al. [10], it allows us to work with the well-established classical ideal functionalities instead of providing an explicit *subversion interface.*

A *speciously subverted code* $\bar{\pi}$ of a protocol between two parties (which we will call Alice and Bob) can informally be described by looking at a protocol, where one party (Alice) might run the speciously subverted code while her partner (Bob) is incorruptible which means that the shell ignores all types of corruption. An environment $\mathcal{Z}$, seen as a *decoder*, *distinguisher*, and *tester*, shall not be able to distinguish between two protocols Π and Π', except with negligible probability. Here, the protocol Π encompasses of the communication between Bob and an incorruptible Alice which runs π following the specification. The other protocol

$\mathit{\Pi}'$ specifies the same protocol as $\mathit{\Pi}$ except that Alice is corruptible and runs the speciously subverted code $\bar{\pi}$. Usually, there is some knowledge shared between the attacker providing the subverted code and the subverted code that is not accessible to the distinguisher, such as cryptographic keys embedded into the subverted code. To model this, we thus assume that that the subverted code $\bar{\pi}$ is equipped with additional information i uniformly drawn from some set I. We assume in the following that there is a PPT algorithm that can sample uniformly from I. The subverted code equipped with this information is denoted by $\bar{\pi}_i$ and the runs of the protocol by $\mathit{\Pi}_i$ or $\mathit{\Pi}'_i$. For the sake of simplicity, we will often also write $\bar{\pi}$ instead of $\bar{\pi}_i$, when the current information is not important to the discussion.

Definition 4 (Speciously subverted code $\bar{\pi}$). *We say, $\bar{\pi}$ is a* speciously subverted code*, if and only if, for all PPT environments $\mathcal{Z}$ it holds that*

$$1 - \Pr_{i \leftarrow \$ I}[EXEC_{\mathit{\Pi}_i,\mathcal{A},\mathcal{Z}}(\kappa, r) \approx EXEC_{\mathit{\Pi}'_i,\mathcal{A},\mathcal{Z}}(\kappa, r)]$$

is negligible, where $\mathcal{A}$ is the dummy adversary and, for sampled information i, $\mathit{\Pi}_i$ and $\mathit{\Pi}'_i$ are the protocols as described before.

Note that the environment here does not have access to the information i, as it only represents the distinguisher here that aims to detect the subversion and not the attacker (that already knows about the subversion).

Valid specious codes. Following this definition, in the corruption model, the shell should only accept *valid specious subversion codes.* Clearly, we cannot enforce arbitrary rules that the subverted code should follow, as no shell might be able to check these rules since many of these rules would lead to undecidable problems. As explained above, in this paper, we only consider attacks that change the randomness, so the shell only accepts subversion codes π which exchange at most the randomness function for an adversarially-chosen function. This function has read-only access to the randomness tape and the whole state of the party including its `sec`. This function will then be inserted by the shell into the body such that each call to a so-called `getRandomness`-function is replaced by a call to this `getRandomness`-function of $\bar{\pi}$. Note that this randomness function can even take the `sec` as in- and output. However, no other change of the code run by the body is allowed. It is easy to see that when restricting the attack to only changing the randomness function, the subverted code fulfils the definition of speciousness. Practically, it makes no difference if the shell only accepts this new randomness function upon invocation or later on. In the latter case, the function is inserted in the body such that all prior access to the randomness tape is replaced by this adversarially-chosen randomness function.

More Details on Corruptions Considered in This Paper We will consider the plain UC model and allow static specious corruptions. Because of the static property, the secret can only be inserted after the corruption step, though it can be updated throughout the protocol run. In general, a party can only be corrupted in one way. This is controlled via a corruption flag that is set if a corruption is valid in place, and subsequent corruption commands are ignored. We omit writing the flags explicitly for better readability.

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 the firewall and "the outside" can be represented by only communicating with one partner. However, the model is also expandable to multi-party computations as we only consider the plain UC model here.

Finally, we only consider attacks captured by the classical UC model. Different attacks using, e.g., the timing behaviour or other physical properties of an implementation are thus out of scope. However, we expect that our model could easily be integrated into different UC extensions capturing such properties such as the model of Canetti et al. [50].

4 Achieving Subversion Resilience with the Help of Reverse Firewalls

In this section, we will formalise how to protect implementations against subversion attacks. 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 [9]. 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.

4.1 Introduction to Reverse Firewalls

After formalising the subversion corruption model and the specious attacks considered in this paper in the previous Section 3, we will now discuss one approach of protecting protocols against such attacks. While different techniques exist in the literature, we will focus on the seminal standard of *reverse firewalls* introduced by Mironov and Stephens-Davidowitz [14]. Informally, such a reverse firewall RF (in short only called firewall or RF) is a device additionally installed by a party $\mathcal{P}$ to *protect* the output of their subverted implementation from *signalling*. The firewall obtains all outgoing communication from $\mathcal{P}$ and all ingoing communication to it. Note that while they are two separate parties, we will call $\mathcal{P}$ a (main) party of a protocol Π and RF the sub-party of $\mathcal{P}$ for easier distinguishability. In this work, we will analyse the security of party $\mathcal{P}$ gained by its RF in the case of subversion corruption. Therefore, we have always 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, whenever $\mathcal{P}$ sends out a message m to another party $\mathcal{P}_j$, the message is first given via an immediate channel to RF, which then delivers a (possibly modified) message m' to $\mathcal{P}_j$, and vice-versa.

4.2 Fundamental Properties of Reverse Firewalls

The firewall itself does not have any secrets, in particular it is not part of the party it is used with and should also not gain information about any secret. It, therefore, can speak about 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 send.

In [14, 15], 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 trough an honest firewall. This is measured by comparing a subverted party together with an honest firewall against an honest party together with an honest firewall.

The authors of [14, 15] 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 only considering specious subversions, hence, the firewalls considered in this paper must only fulfil the 'weak' properties.

In [14, 15], 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 measured by comparing an honest party together with an honest firewall against an honest party without firewall.

In this work, we will call their 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. In the model of Chakraborty et al. [10], the parties using the firewalls need to be aware of their existence, as they might, e.g., obtain additional information from these firewalls. The notion of stackability is never discussed by Chakraborty et al., but one could, e.g., expect that the party needs to obtain these additional information from every firewall. However, this would only provide stackability, but *not* inner transparency.

4.3 Handling of Reverse Firewalls in UC

As described above, in contrast to Chakraborty et al. [10], we consider subversions that change an input of the underlying algorithm to be suspicious. Hence, 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.

A firewall RF is modelled as a sub-party of main party $\mathcal{P}$ in the real world as visualised in Figure 4. 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.

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 $\mathcal{F}_{\text{AUTH}}$, where 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 Section 8. 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. $\mathcal{Z}$ 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 [10] in that, there, $\mathcal{F}$ 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 [10] 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.

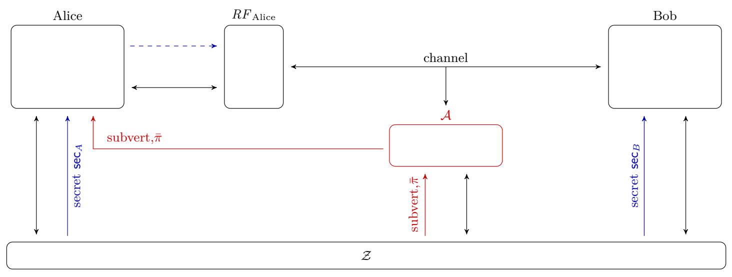


Fig. 4: Informal visualisation of firewalls in the UC model. Information about the secret might be leaked to the firewall (represented by $\dashrightarrow$), however, it is the goal of the firewall to output messages to the channel that no longer contain this leakage. The leakage of Bob through the channel is omitted for simplicity.

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 does not change the underlying functionality. As already discussed above, in [10], 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.

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 [10], 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.

4.4 Modelling of Properties of Reverse Firewalls in UC

While the properties *functionality maintaining*, *security preserving*, and *exfiltration resistance* explained above are clearly needed for the reverse firewalls to work at all, there are a number of 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 [14] in the UC-setting. Furthermore, as we will see, they also allow to simplify the security analysis significantly, as we do not need to to "reprove" the UC-simulatability.

Transparency Firstly, there is the notion of *transparency.* While partially already mentioned by [14], we extend on this 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 5. *(Transparency)* Let $\mathcal{P}$ be a party and RF its firewall and the other party in the protocol be incorruptible. Let Π be the protocol run by $\hat{\mathcal{P}}$ while a (modified) protocol Π' is run by the composed party $\hat{RF} \circ \hat{\mathcal{P}}$ where $\hat{\mathcal{P}}$ and $\hat{RF}$ are the incorruptible versions of $\mathcal{P}$ and RF, respectively. If for all PPT environments $\mathcal{Z}$ it holds that $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r) \approx \text{EXEC}_{\Pi',\mathcal{A},\mathcal{Z}}(\kappa, r)$ where $\mathcal{A}$ is the dummy adversary, then RF provides *outer transparency* for Π' with regard to Π. 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 $\hat{RF} \circ (\hat{RF} \circ \hat{\mathcal{P}})$. Illustrated in Figure 5, neither $\hat{\mathcal{P}}_1$ nor $\hat{\mathcal{P}}_2$ can distinguish, whether $\hat{\mathcal{P}}_1$ uses a firewall. This directly implies functionality maintenance.

To allow an extra communication feedback channel between the firewall and its party, we can weaken this definition to only "outer transparency". Here, Π' can be slightly changed to accommodate the extra information given by its firewall(s). However, as discussed above, this channel can negatively impact the security guarantees and we only construct firewalls with strong transparency.

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.

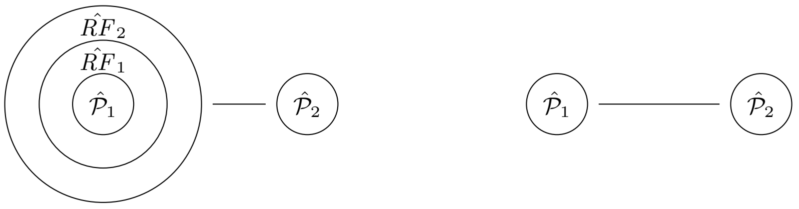


Fig. 5: If both firewalls $\hat{RF}_1$ and $\hat{RF}_2$ are strongly transparent, these two interactions cannot be distinguished.

In [10], 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 above, that would disallow the usage of feedback channels.

Non-Signalling Composed Party Another property to model is the notion that the firewall counteracts the leakage (signal) 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* (it was previously called *exfiltration-resistant* by Mironov and Stephens-Davidowitz [14] and *strong sanitation* by Chakraborty et al. [10]). The main difference to the definition of a specious subversion is that the adversary $\mathcal{A}$ here learns the shared information $i \in I$ contained in the subverted code $\bar{\pi}_i$. We denote that $\mathcal{A}$ is given these information by $\mathcal{A}(i)$.

Definition 6. *(Anti-Signalling)* Let $\mathcal{P}$ be a party and RF its firewall. Let Π be the protocol run by the composed party $\hat{RF} \circ \bar{\mathcal{P}}$ and Π' be the exact same protocol except that it is run by party $\hat{RF} \circ \hat{\mathcal{P}}$ where $\hat{\mathcal{P}}, \hat{RF}$ are the incorruptible versions of $\mathcal{P}$ and RF. Recall that the subverted party $\bar{\mathcal{P}}$ also includes the case that $\bar{\mathcal{P}} = \mathcal{P}$. The firewall RF is anti-signalling if for all PPT environments $\mathcal{Z}$ it holds that

$$1 - \Pr_{i \leftarrow \$ I}[\mathrm{EXEC}_{\Pi,\mathcal{A}(i),\mathcal{Z}}(\kappa, r) \approx \mathrm{EXEC}_{\Pi',\mathcal{A}(i),\mathcal{Z}}(\kappa, r)],$$

is negligible, where $\mathcal{A}_i$ is the dummy adversary having access to information i.

In most applications, the firewalls even guarantee *perfect* anti-signalling, where $\mathrm{EXEC}_{\Pi,\mathcal{A}_i,\mathcal{Z}}(\kappa, r)$ and $\mathrm{EXEC}_{\Pi',\mathcal{A}_i,\mathcal{Z}}(\kappa, r)$ are distributed identically. The definition of anti-signalling is visualised in Figure 6.

Note that a party that is subverted due to a specious subversion might very well be signalling, since the environments are given different levels of knowledge:

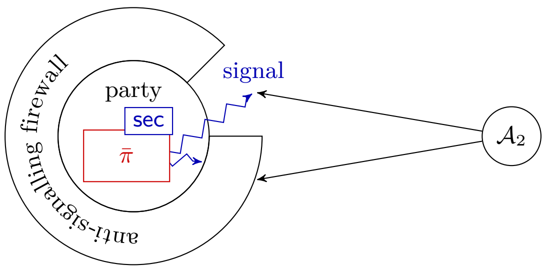


Fig. 6: Informal visualisation of an anti-signalling firewall, which can make the composed party non-signalling even though the party itself is signalling.

In the definition of speciousness, the adversary (and therefore the environment) is not provided with the used information i. In contrast, non-signalling requires indistinguishability even when the adversary learns such an information.

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 (nearly) indistinguishable from the interaction of two honest parties. First, the notion of anti-signalling allows to replace the subverted party using the honest firewall by 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.

5 Security Analysis of Subversion Corrupted Protocols with Reverse Firewalls

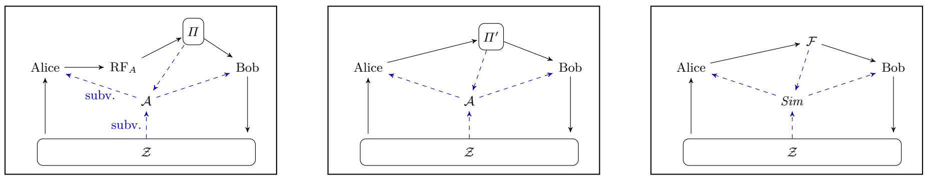
In this section, we want to show how one analyses the security of a protocol susceptible to specious subversion corruptions that is at the same time protected by firewalls. Therefore, we combine the modelling results of Section 3 to expand a standard protocol to subversion corruption with the results of Section 4 to analyse which additional cases the standard protocol can be protected against. As before in this work, we will use the running example of a commitment protocol between Alice with its firewall and Bob. Detailed information of the commitment protocol is given in Section 6.

We want to prove that a protocol $\Pi \geq_{\mathrm{UC}} \mathcal{F}$ under subversion corruption with specious codes. To do so we perform the following steps.

1. We assume, that in the literature, there is a protocol Π' ((b) in Figure 7) given which UC-realises the desired ideal functionality $\mathcal{F}$ ((c) in Figure 7). This will be our *base protocol* Π'.
2. On the one hand, we adapt the base protocol to allow subversion corruption by expanding the corruption model as described in Section 3. On the other hand, we add firewalls to the protocol as described in Section 4. The latter models a protection strategy against subversion corruption. These protocol adaptions yields the protocol Π ((a) in Figure 7).

3. 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 2.
4. From the transitivity of UC-emulation [29, Sec. 4.3] follows that if $\Pi \geq_{\text{UC}} \Pi'$ and $\Pi' \geq_{\text{UC}} \mathcal{F}$, then $\Pi \geq_{\text{UC}} \mathcal{F}$.

Proceeding this way reduces the overhead of proving that $\Pi \geq_{\text{UC}} \mathcal{F}$ under party-wise subversion, semi-honest, and malicious corruption by hand in one step.



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

Fig. 7: The adapted protocol (a) of the base protocol (b) provides modelling of subversion corruption and reverse firewalls. The base protocol (b) UC-realises the ideal functionality in (c).

First. We assume that there exists a classic protocol Π' which is the base protocol of Π, meaning that there are no firewalls yet included as one does not analyse the resistance against subversion corruptions. This protocol fulfils a certain set of security properties in the way that $\Pi' \geq_{\text{UC}} \mathcal{F}$ under (semi-honest and) malicious corruption.

Second. We adapt Π' such that we can analyse its security under subversion corruption. All adaptions follow the model of Section 3. To begin with, we give every party an interface to get a secret sec from the environment. Note that the secret could also already be in the state of the party. However, this visualises the goal of a subversion corruption to leak any information of some inner secret of that party. By giving the adversary the power to set the secret we model the adversary even more powerful.

Given a $\Pi'_{\text{COM}} \geq_{\text{UC}} \mathcal{F}_{\text{COM}}$ in the $\{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{AUTH}}\}$-hybrid model under malicious corruption. We model Π_{COM} where Alice and Bob receive a symbolic secret and a firewall which rerandomises the communication. Note that since this is a non-interactive commitment, only Alice needs a firewall which rerandomises the commitment $c' = c \cdot \text{COM}(0, r \overset{\$}{\leftarrow} \mathbb{Z}_p)$ and later opens to the message with the rerandomised decommitment. The environment gets the additional option to subversion-corrupt Alice with a specious code, i.e., an adversarially-chosen randomness function.

Third. As a protection against subversion corruptions, we will analyse firewalls and, therefore, give every main party $\mathcal{P}$ which could leak something about its secret via subversion during the protocol run a firewall RF following the modelling of Section 4.3. A firewall RF is a party on its own and thereby the sub-party to its main party $\mathcal{P}$.

Fourth. To show the security after these changes, we will make use of the transitivity of UC-emulation [29, Sec. 4.3], meaning that if $A \geq_{\mathrm{UC}} B$ and $B \geq_{\mathrm{UC}} C$, then $A \geq_{\mathrm{UC}} C$.

Theorem 1 (informal). *Given a protocol Π'. Expanding the corruption behaviour to allow specious subversion and add reverse firewalls as protection layer to the protocol. We show that the adapted protocol Π UC emulates the protocol Π'. Therefore, the protocol Π is party-wise secure under the corruptions of Π' and specious subversion.*

To prove Theorem 1, we provide an overview of the corruption behaviour in Table 2 for the corruption transition of a main party $\mathcal{P}$ and its sub-party RF in Π to its composed classic corruption behaviour in Π'. We want to arrive at the classic corruptions of semi-honest and malicious corruption to be able to make use of the transitivity of UC-emulation. This table gives also a good understanding of how a firewall RF affects a protocol and how it derives subversion-resilience. We require that the used RF fulfils the definition of transparency and anti-signalling as defined in Section 4.4. To analyse the security of a protocol with firewalls, one can observe the influence of the sub-party RF of the respective main party $\mathcal{P}$.

Party	**Firewall**	**composed classical behaviour**
honest	honest	honest
honest	semi-honest	honest
honest	malicious	malicious/honest, see Section 8
malicious	honest/semi-honest/malicious	malicious
subversion	honest	honest
subversion	semi-honest	semi-honest
subversion	malicious	malicious

Table 2: Different corruption combinations and their composed classical corruption behaviour. The lines in grey differ from the table given in [10].

In some transition cases, one could see the composed classical behaviour as over-approximation of the actual behaviour, i.e., some security guarantees which are expected in Π are not given in Π', e.g. a honest or subverted $\mathcal{P}$ would not loose the binding property of its commitment in Π but the composed corruption behaviour in Π' would give the content of the commitment to the adversary. We show what is achievable with the notions of UC-simulation, that not all securities guarantees are captured will be discussed further in Section 8.4. The notable achievable double state of "honest/malicious" for an honest $\mathcal{P}$ and a malicious RF is the result of an important insight: If we do not want to adapt given protocols in a complex way, we can only hope for the composed classical behaviour of a malicious party. But the main aspect why it behaves maliciously in the natural understanding is that RF can impersonate $\mathcal{P}$. This is naturally given in the composed behaviour, where $\mathcal{A}$ can impersonate $\mathcal{P}$. In Section 8, we present several ways to circumvent the impersonate attack vector of RF. If we add one of the solutions to the protocol, we can achieve and prove via UC-simulation that the composed party behaves honest.

Since we only consider static corruption, we can provide a simulator consisting of a concatenation of all case-wise simulators. Note that *Sim* informs the corruption aggregation ITI in the way of the transition table, meaning that it reports the corruption state of the whole composed party together. That this is equivalent to the report in the real world is given by the modelling described in Section 3.

E.g., if a party $\mathcal{P}$ is subverted and its firewall is malicious, the corruption aggregation ITI only reports that $\mathcal{P}$ is malicious. Otherwise $\mathcal{Z}$ would be able to tell the worlds apart by noticing e.g. that there exists RF.

Theorem 2. *Given protocol Π with main parties $\mathcal{P}_i$ under party-wise specious subversion, semi-honest, and malicious corruption, and the (sub-)parties RF_i under party-wise semi-honest and malicious corruption in the $\{\mathcal{F}_{CRS}, \mathcal{F}_{AUTH}\}$-hybrid model. Given protocol Π' 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 behaviour of the parties $RF_i \circ \mathcal{P}_i$ transition from their individual party-wise corruption to their composed classical corruption behaviour following the corruption transition table Table 2. Let RF_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.*

For the most interesting cases of a subversion corrupted parties, we will provide a simulator.

We only analyse the behaviour of the composed party, therefore the simulator does only provide simulation until the network, which starts with $\mathcal{F}_{\text{AUTH}}$ and goes on with communication to further parties. This is no limitation since the simulator can simulate the network as in the base protocol Π'.

$Sim_{\Pi'}$

$Sim_{\Pi'}$ simulates for a protocol Π' with main party $\mathcal{P}_{\Pi'}$ under static semi-honest and malicious corruption the protocol Π with main party $\mathcal{P}$ under static subversion, semi-honest, or malicious corruption, and its sub-party RF under static semi-honest or malicious corruption. A "network" is used to represent the in- and output interface of the (composed) $\mathcal{P}$, i.e. $\mathcal{F}_{\text{AUTH}}$ and all further protocol part.

Behaviour:

\\\\ Case: Subverted party $\mathcal{P}$ and honest firewall RF:

- Upon receiving $(\texttt{corrupt}, sid, \mathcal{P}, (\texttt{subversion}, \overline{\pi}))$ from $\mathcal{Z}$, store $(sid, \mathcal{P}, \texttt{subversion}, \overline{\pi})$. and ignore further corrupt messages.
- On all other backdoor messages, forward them to the respective recipient.

\\\\ Since firewall honest, there are no backdoor messages between firewall and $\mathcal{Z}$.

\\\\ Case: Subverted party $\mathcal{P}$ and semi-honest firewall RF:

- Upon receiving $(\texttt{corrupt}, sid, \mathcal{P}, (\texttt{subversion}, \overline{\pi}))$ and $(\texttt{corrupt}, sid, RF, \texttt{semi-honest})$ as the first two messages from $\mathcal{Z}$,
  store $(sid, \mathcal{P}, \texttt{subversion}, \overline{\pi})$,
  send backdoor message $(\texttt{corrupt}, sid, \mathcal{P}_{\Pi'}, \texttt{semi-honest})$ to $\mathcal{P}_{\Pi'}$,
  Ignore further corrupt messages.
- Upon receiving backdoor messages $(\texttt{semi-honest}, sid, mid, \mathsf{state})$ from $\mathcal{P}_{\Pi'}$, simulate the subversion corruption with $\overline{\pi}$ and the firewall on state in-the-head, and output the updated state $(\texttt{semi-honest}, sid, mid, \mathsf{state}')$ in the name of RF to $\mathcal{Z}$.
- On "continue" from $\mathcal{Z}$ to RF, send "continue" to $\mathcal{P}_{\Pi'}$.
- On all other backdoor messages, forward them to the respective recipient.

\\\\ Case: Subverted party $\mathcal{P}$ and malicious firewall RF:

- Upon receiving $(\texttt{corrupt}, sid, \mathcal{P}, (\texttt{subversion}, \overline{\pi})$ and $(\texttt{corrupt}, sid, RF, \texttt{malicious})$ as the first two messages from $\mathcal{Z}$,
  store $(sid, \mathcal{P}, \texttt{subversion}, \overline{\pi})$,
  send backdoor message $(\texttt{corrupt}, sid, \mathcal{P}, \texttt{malicious})$ to $\mathcal{P}$,
  and receive all in-/output messages from/to $\mathcal{Z}$ to/from $\mathcal{P}$.
  Ignore further corrupt messages.
- On input from $\mathcal{Z}$ to $\mathcal{P}$,
  run in-the-head $\mathcal{P}$'s $\overline{\pi}$ and send the output in the name of RF to $\mathcal{Z}$.
- On input in name of RF to the network, forward it to the network as if $\mathcal{P}$ would be malicious.
- On output from the network to $\mathcal{P}$, forward it as input of RF to $\mathcal{Z}$.
- On output in the name of RF to $\mathcal{P}$, run in-the-head $\mathcal{P}$'s $\overline{\pi}$ and send the output $\overline{\pi}$ would send in the name of $\mathcal{P}$.

Proof Sketch of Theorem 2. To prove that Π UC-emulates Π', we have to show that it is indistinguishable for $\mathcal{Z}$ which protocol is executed. $Sim_{\Pi'}$ simulates the (corruption) behaviour of the main party $\mathcal{P}$ and its sub-party RF in Π to be indistinguishable from the (corruption) behaviour given by Table 2 of the main party $\mathcal{P}$ in Π'. The simulator only captures the communication flow till and from the network as after that, it works like the dummy adversary in Π'. We assume that the network starts with $\mathcal{F}_{\mathrm{AUTH}}$ as we analyse protocols in the $\{\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{AUTH}}\}$-hybrid model. Remark that it does not matter how the other party (or parties) are corrupted in the protocol and whether they have firewalls installed, because the simulator can be used party-wise and combined modularly. Hence, we can focus on the two-party case with the view on one main party $\mathcal{P}$ and its sub-party RF which can be extended modularly to more parties. We make use of $Sim_{\Pi'}$ for all cases considering a subverted party. The other cases are directly discussed in the following proof sketch, such that we provide an argumentation for every line, i.e., all possible cases given the static corruption combinations, in Table 2.

Honest $\mathcal{P}$ and honest RF behave indistinguishable from honest $\mathcal{P}_{\Pi'}$. If RF is strongly transparent, $\mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r) \approx \mathrm{EXEC}_{\Pi',Sim,\mathcal{Z}}(\kappa, r)$, where both the main party in Π and its RF is honest, and the main party in Π' is honest as well. Hence, it has no influence on the simulation. The distributions are identically and independently distributed on both sides. Therefore, the simulator can just ignore the firewall as no secret will be leaked. This is a special case of the variant where $\mathcal{P}$ is subverted.

Honest $\mathcal{P}$ and semi-honest RF behave indistinguishable from honest $\mathcal{P}_{\Pi'}$. From the analysis above, we know that an honest party together with an honest, strongly-transparent firewall is indistinguishable from honest $\mathcal{P}_{\Pi'}$. When now replacing the firewall with the semi-honestly corrupted RF, the composed party *behaves* the same as with the honest firewall. The only difference between the composed parties is that now, the adversary additionally learns about its state. However, since the party is honest, no secret will be leaked to the firewall (and therefore through its state). Consequently, we can see the composed party as honest. The simulator has to generate the state and output of RF by running the code on the output of $\mathcal{P}$ which the simulator simulates during the network communication part.

Honest $\mathcal{P}$ and malicious RF behave indistinguishable from honest/malicious $\mathcal{P}_{\Pi'}$. This case is an interesting edge-case as the firewall is maliciously corrupted but the party itself is honest. Because of the latter, no secret will be leaked, which in turn means that the firewall can leak nothing. However, the firewall communicates in the name of $\mathcal{P}$ and, therefore, can impersonate it by generating its own messages. There are different approaches to handle this situation: The worst-case consideration would be viewing the composed party as maliciously corrupted. In this case, the simulator could simulate the behaviour of honest $\mathcal{P}$ and malicious RF. Remark that Canetti models malicious corruption in a way that “in an activation due to an incoming input or subroutine-output, the shell sends the entire local state to the adversary” [29, §7.1.1 Byzantine corruption, line 6f]

This modelling also captures what one would expect by maliciously corrupting a party —the input to this party goes to the adversary. Therefore, *Sim* receives the input from $\mathcal{Z}$, which gives *Sim* the power to run $\mathcal{P}$'s code in the head on this input to simulate an honest output for the RF. Running RF's code in-the-head on output to $\mathcal{P}$ from $\mathcal{Z}$ completes the simulation and both sides are indistinguishable. However, malicious corruption behaviour would imply that none of the security properties is guaranteed any more, even though this is generally not the case. To get the composed party to behave like an honest party, one has to circumvent impersonation attacks of RF. This is captured with modelling and more practical solutions discussed below in Section 8.

Malicious $\mathcal{P}$ and honest/semi-honest/malicious RF behave indistinguishable from malicious $\mathcal{P}_{\Pi'}$. The two executions are indistinguishable as *Sim* gets all in-/output to/from the malicious $\mathcal{P}$ and can run RF's code in-the-head before sending it to $\mathcal{Z}$ or network, respectively. In the case where RF is semi-honest, output the produced state of RF to $\mathcal{Z}$ and continue on "ok" from $\mathcal{Z}$. In the case where RF is malicious, simulate communication between $\mathcal{P}$ and RF as between two malicious parties, i.e., send each in-/output to $\mathcal{Z}$.

Remark 2. Following Section 2.3.3 in [51], we know that the UC-simulation under malicious corruption implies the UC-simulation under augmented semi-honest corruption. This means that if we can show that the base protocol is secure if $\mathcal{P}_{\Pi'}$ is malicious, it is also secure if $\mathcal{P}_{\Pi'}$ is augmented semi-honest. Again, if the base protocol Π' is also secure against semi-honest corruptions, this type of corruption can also be considered for the composed parties.

Subverted $\mathcal{P}$ and honest RF behave indistinguishable from honest $\mathcal{P}_{\Pi'}$. If RF is anti-signalling, $\mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, r) \approx \mathrm{EXEC}_{\Pi',Sim_{\Pi'},\mathcal{Z}}(\kappa, r)$, where the main party in Π is subversion corrupted and its RF honest, and the main party in Π is honest. In the case of the party $\mathcal{P}$ being subverted and its firewall RF being honest, since the firewall is anti-signalling, the composed party is non-signalling and, hence, subversion resilient. This means that the party cannot leak the secret speciously via the protocol.

Subverted $\mathcal{P}$ and semi-honest RF behave indistinguishable from semi-honest $\mathcal{P}_{\Pi'}$. The two executions are indistinguishable as *Sim* can produce the state of the semi-honestly corrupted RF as by running its code in the head after applying $\overline{\pi}$ to $\mathcal{P}_{\Pi'}$'s state. Since a semi-honest firewall still follows its code, any leakage output through the subverted $\mathcal{P}$ will be removed, but the secret sec would be leaked via the state. Note that while the executions are indistinguishable, we can simulate this case, though the protocol is not subversion resilient any more.

Subverted $\mathcal{P}$ and malicious RF behave indistinguishable from malicious $\mathcal{P}_{\Pi'}$. This case works similar to the case of an honest $\mathcal{P}$ and a malicious RF. In this case, *Sim* also receives the secret as an input to $\mathcal{P}$ and can, therefore, simulate as well that the secret is leaked. As $Sim_{\Pi'}$ knows the in-/output of $\mathcal{P}$ and RF, it can easily simulate running $\bar{\pi}$ and RF in-the-head if needed. Hence, both sides are indistinguishable. One cannot hope to achieve an honest behaviour for the composed party as the secret is leaked by the malicious firewall, but one can reduce the attack vector by impersonation attacks of RF, as discussed in Section 8.

Remark 3. If the party is subverted, we can view the composed party of a subverted party and its corrupted firewall as a corrupted party. This might seem like an over-approximation, however, in the worst case, a subversion could leak the seed of the whole state of $\mathcal{P}$ which would then be equal to the direct semi-honest corruption of $\mathcal{P}$. Hence, the composed party of a subverted $\mathcal{P}$ with a corrupted RF behaves, in the worst case, the same as a corrupted $\mathcal{P}$. However, we can show that a base protocol to which firewalls have been added still simulates the ideal functionality of the base protocol. Even though in the case of an honest party and a malicious firewall, one needs to take care of impersonation attacks by the firewall (see Section 8), if we add protection against such impersonation attacks of the firewall, we can simulate the behaviour of a subverted party and its malicious firewall to a semi-honest party.

Corollary 1. *Given Theorem 2, it follows that $\Pi \geq_{UC} \mathcal{F}$.*

From the transitivity of UC-emulation [29] and Theorem 2 it follows directly that $\Pi \geq_{\mathrm{UC}} \mathcal{F}$ because $\Pi \geq_{\mathrm{UC}} \Pi'$ and $\Pi' \geq_{\mathrm{UC}} \mathcal{F}$.

Applying Our Model After having proved that we can collapse a party and its firewall into a composed classical party, we can provide a short checklist how to prove the security of the overall protocol when using firewalls.

Given a base protocol Π' UC-realising an ideal functionality $\mathcal{F}$ and secure and semi-honest and malicious corruption. To analyse the resilience against specious subversions via cryptographic reverse firewalls, one only has to proceed the following steps:

1. Adapt the protocol according to Section 3 to expand the corruption factor by specious subversion,
2. add reverse firewalls to the protocol as described in Section 4,
3. show that the new protocol Π is correct,
4. and that the firewalls are both strongly transparent and anti-signalling.

Following Theorem 2, the new protocol Π UC emulates the base protocol Π' which UC-realises the ideal functionality in the $\{\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{AUTH}}\}$-hybrid model, where the main parties are malicious, subversion and possibly semi-honest corrupted with a specious code $\bar{\pi}$, and the respective sub-parties *RF* are honest, semi-honest or malicious. From transitivity of UC emulation follows that Π also UC-realises the ideal functionality $\mathcal{F}$ in the $\{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{AUTH}}\}$-hybrid model.

Remark 4. Even though Theorem 2 is also applicable for maliciously corrupted firewalls, as discussed in Section 8, additional measures have to be taken to protect against impersonation attacks.

6 Commitments

The considered commitment scheme based on the one in [30] is a transformation of the oblivious transfer scheme presented below in Section 7 by the same authors. Additionally, it is also equal to the bit commitment version of one discussed in [10] if there, one would set the input of the firewall to zero. However, in contrast to [10], we can directly reuse the security analysis of [30] and do not need to "reprove" the UC-security of the protocol.

Since the commitment scheme has already been widely discussed throughout this paper, we only give a short description by illustrating the considered protocol in Figure 8. This protocol can be shown to fulfil the following:

Theorem 3. *The protocol Π_{srCOM} UC-realises the functionality $\mathcal{F}_{COM}$ in the $\{\mathcal{F}_{CRS}, \mathcal{F}_{AUTH}\}$-hybrid model, where the main parties are malicious, semi-honest or subversion corrupted with a specious code $\bar{\pi}$, and the respective sub-parties RF are honest, semi-honest or malicious.*

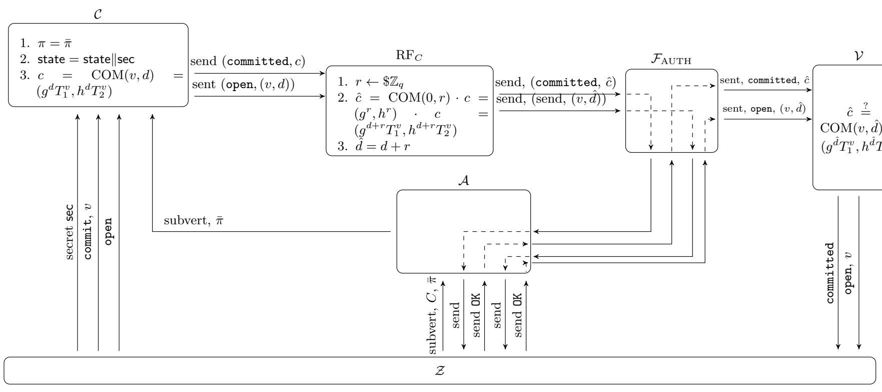

$\mathcal{C}$
1. $\pi = \bar{\pi}$
2. state = state||sec
3. $c = \text{COM}(v,d) = (g^d T_1^v, h^d T_2^v)$
send (committed, c)
sent (open, (v,d))
RF$_C$
1. $r \leftarrow \$\mathbb{Z}_q$
2. $\hat{c} = \text{COM}(0,r) \cdot c = (g^r, h^r) \cdot c = (g^{d+r}T_1^v, h^{d+r}T_2^v)$
3. $\hat{d} = d + r$
send, (committed, $\hat{c}$)
send, (send, $(v, \hat{d})$)
$\mathcal{F}_{\text{AUTH}}$
sent, committed, $\hat{c}$
sent, open, $(v, \hat{d})$
$\mathcal{V}$
$\hat{c} \overset{?}{=} \text{COM}(v, \hat{d}) = (g^{\hat{d}}T_1^v, h^{\hat{d}}T_2^v)$
$\mathcal{A}$
subvert, $\bar{\pi}$
secret sec
commit, v
open
subvert, C, $\bar{\pi}$
send
send OK
send
send OK
committed
open, v
$\mathcal{Z}$

Fig. 8: Real run of the commitment scheme (pictured without $\mathcal{F}_{\text{CRS}}$).

6.1 Ideal Functionality: Commitment

$\mathcal{F}_{\text{COM}}$

Two-party ideal bit commitment protocol between a committer $\mathcal{C}$ and a verifier $\mathcal{V}$.

- Upon receiving an input $(\texttt{commit}, sid, v)$ from $\mathcal{C}$, where $v \in \{0, 1\}$, record the tuple (sid, v) and generate a public delayed output $(\texttt{committed}, sid)$ to $\mathcal{V}$. Ignore subsequent commands of the form $(\texttt{commit}, sid, \cdot)$.
- Upon receiving an input $(\texttt{open}, sid)$ from $\mathcal{C}$, generate a public delayed output $(\texttt{open}, sid, v)$ to $\mathcal{V}$. Ignore subsequent commands of the form $(\texttt{open}, sid, \cdot)$.
  \\\\ Corruption Handling:
- On input $(\texttt{secret}, sid, \mathcal{P}, \mathsf{sec})$, store $(\texttt{secret}, sid, \mathcal{P}, \mathsf{sec})$.
  If RF is corrupted send $(\texttt{secret}, sid, \mathcal{P}, \mathsf{sec})$ to Sim.
- Ignore all further inputs.

The ideal run of the commitment scheme is illustrated in Figure 9.

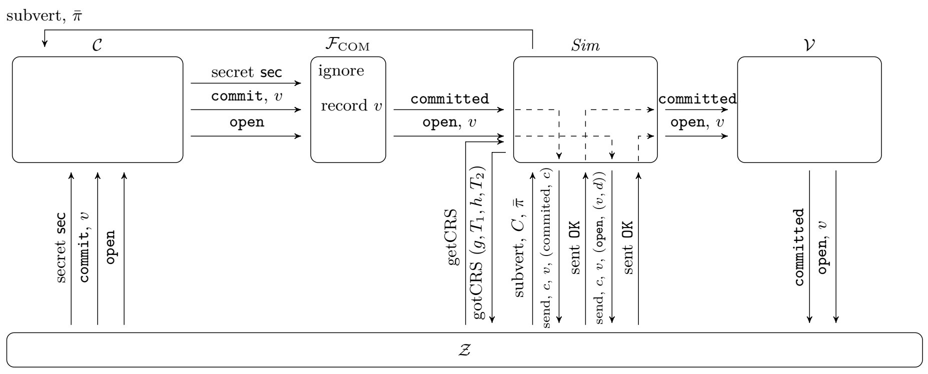


Fig. 9: Ideal run of the commitment scheme

6.2 Real Protocol: Subversion-resilient Commitment based on DLog [30]

As was done in [10], who base their commitment on the same scheme by [30], the protocol is in a modified common reference string (CRS) model where the first three elements are assumed to be random. The group $\mathbb{G}$ with generator g and the field $\mathbb{Z}_q$ are public inputs.

The base commitment scheme in [30] proves security under malicious corruption. Again, this implies security under augmented semi-honest corruption. Plain semi-honest security, however, has not been shown yet. If one cares or doubt about this, we leave this as an open problem to the reader, as this would have to be proven first for the case above in Theorem 2 where the subverted party and plain semi-honest corrupted firewall behave indistinguishable from a plain semi-honest party.

$$\Pi_{\mathbf{srCOM}}^{\mathcal{F}_{\mathbf{CRS}},\mathcal{F}_{\mathbf{AUTH}}}$$

Realises:
Subversion-resilient two-party bit commitment between a main party committer $\mathcal{C}$, its sub-party firewall $RF_{\mathcal{C}}$, a main party receiver $\mathcal{R}$, and an adversary $\mathcal{A}$.

Parameters:

- Functionality $\mathcal{F}_{\text{AUTH}}$.
- Functionality $\mathcal{F}_{\text{CRS}}$:
- Only static subversion corruption is pictured, others omitted.
- Every main-party has an empty initialised secret tape.

Behaviour of Party $\mathcal{C}$:

\\ Subversion Corruption Handling:

- Upon receiving $(\texttt{corrupt}, sid, (\texttt{subversion}, \bar{\pi}))$ as first message after invocation from $\mathcal{A}$,
  shell checks if $\bar{\pi}$ is a valid specious code by the criteria given in *Section* 3.2, that is that $\bar{\pi}$ only consists of a randomness function.
  If $\bar{\pi}$ is valid specious, shell replaces all randomness function in the body with the one from $\bar{\pi}$, and sets a subversion corruption flag.
  Else ignore.
  Ignore subsequent corruption commands $(\texttt{corrupt}, sid, \cdot)$.
- Upon receiving $(\texttt{secret}, sid, s)$ from $\mathcal{Z}$, write s on the secret tape.

\\ Commitment Handling:

- Upon receiving $(\texttt{commit}, sid, v)$ from $\mathcal{Z}$, where $v \in \{0,1\}$, send $(\texttt{value}, sid)$ to $\mathcal{F}_{\text{CRS}}$.
- Upon receiving $(\texttt{crs}, sid, (g, T_1, h, T_2))$ from $\mathcal{F}_{\text{CRS}}$:
  get d from the randomness function, \\ presumably subverted
  and set $c := (c_0, c_1) = \texttt{Com}(v, d) = (g^d \cdot T_1^v, h^d \cdot T_2^v)$.
  Send $(\texttt{send}, sid, \mathcal{C}, \mathcal{V}, (\texttt{commit}, c))$ to $RF_{\mathcal{C}}$ via an immediate channel.
- Upon receiving $(\texttt{open}, sid)$ from $\mathcal{Z}$, output $(\texttt{send}, sid, \mathcal{C}, \mathcal{V}, (\texttt{opened}, (v, d))$ to $RF_{\mathcal{C}}$.

Behaviour of Sub-party $RF_{\mathcal{C}}$:

- Upon receiving $(\texttt{send}, sid, \mathcal{C}, \mathcal{V}, (\texttt{commit}, c))$ from $\mathcal{C}$, send $(\texttt{value}, sid)$ to $\mathcal{F}_{\text{CRS}}$.
- Upon receiving $(\texttt{crs}, sid, (g, T_1, h, T_2))$ from $\mathcal{F}_{\text{CRS}}$, sample $r \overset{\$}{\leftarrow} \mathbb{Z}_q$, and set $c' := (c_0', c_1') = c \cdot \texttt{Com}(0, r) = (c_0 \cdot g^r, c_1 \cdot h^r)$.
  Send $(\texttt{send}, sid, \mathcal{C}, \mathcal{V}, (\texttt{committed}, c'))$ in the name of $\mathcal{C}$ to $\mathcal{F}_{\text{AUTH}}$.
- Upon receiving $(\texttt{send}, sid, \mathcal{C}, \mathcal{V}, (\texttt{opened}, (v, d)))$ from $\mathcal{C}$, set $d' := d + r$, and send $(\texttt{send}, sid, \mathcal{C}, \mathcal{V}, (\texttt{opened}, (v, d')))$ to $\mathcal{F}_{\text{AUTH}}$.

Behaviour of Party $\mathcal{V}$:

\\ Subversion Corruption Handling:

- Upon receiving $(\texttt{corrupt}, sid, (\texttt{subversion}, \bar{\pi}))$ as first message after invocation from $\mathcal{A}$,

shell checks if $\bar{\pi}$ is a valid specious code by the criteria given in *Section* 3.2, that is that $\bar{\pi}$ only consists of a randomness function.
If $\bar{\pi}$ is valid specious, shell replaces all randomness function in the body with the one from $\bar{\pi}$, and sets a subversion corruption flag.
Else ignore.
Ignore subsequent corruption commands $(\texttt{corrupt}, sid, \cdot)$.

- Upon receiving $(\texttt{sent}, sid, \mathcal{C}, \mathcal{V}, c')$ from $\mathcal{F}_{\text{AUTH}}$, output $(\texttt{committed}, sid)$ to $\mathcal{Z}$.
- Upon receiving $(\texttt{sent}, sid, \mathcal{C}, \mathcal{V}, (v, d'))$ from $\mathcal{F}_{\text{AUTH}}$, output $(\texttt{opened}, sid, v)$ to $\mathcal{Z}$.

Behaviour of Party $\mathcal{A}$:

\\\\ Subversion Corruption Handling:

- Upon receiving input $(\texttt{corrupt}, sid, \mathcal{C}, (\texttt{subversion}, \bar{\pi}))$ from $\mathcal{Z}$, send $(\texttt{corrupt}, sid, (\texttt{subversion}, \bar{\pi}))$ to backdoor tape of $\mathcal{C}$.

\\\\ CRS Handling:

- Upon receiving input $(\texttt{value}, sid)$ from $\mathcal{Z}$, forward $(\texttt{value}, sid)$ to $\mathcal{F}_{\text{CRS}}$.
- Upon receiving input $(\texttt{crs}, sid, (g, T_1, h, T_2))$ from $\mathcal{F}_{\text{CRS}}$, output $(\texttt{crs}, sid, (g, T_1, h, T_2))$ to $\mathcal{Z}$.

\\\\ AUTH Handling:

- Upon receiving input $(\texttt{send}, sid, \mathcal{C}, \mathcal{V}, msg)$ from $\mathcal{F}_{\text{AUTH}}$, output $(\texttt{send}, sid, \mathcal{C}, \mathcal{V}, msg)$ to $\mathcal{Z}$.
- Upon receiving input $(\texttt{sendok}, sid)$ from $\mathcal{Z}$, forward $(\texttt{sendok}, sid)$ to $\mathcal{F}_{\text{AUTH}}$.

The real run of the commitment scheme is illustrated in Figure 8.

6.3 Security Proof of Theorem 3

Proof. Since the firewall only rerandomises the used randomness, one can easily see that Π_{srCOM} is correct as is shown in Lemma 1.

Lemma 1. *Π_{srCOM} is correct.*

Proof. Π_{srCOM} is correct if the verifier $\mathcal{V}$ opens the received commitment c' under the received decommitment information d' to committed-to value v of the sender S, therefore $\mathcal{V}$ checks that $c' = \text{COM}(v, d')$, with $d' = r + d$ set from RF. It holds that

$$\begin{aligned}
c' &= \text{COM}(0, r) \cdot c \\
&= \text{COM}(0, r) \cdot \text{COM}(v, d) \\
&= (g^r \cdot T_1^0, h^r \cdot T_2^0) \cdot (g^d \cdot T_1^v, h^d \cdot T_2^v) \\
&= (g^r \cdot (g^d \cdot T_1^v), h^r \cdot (h^d \cdot T_2^v)) \\
&= (g^{r+d} \cdot T_1^v, h^{r+d} \cdot T_2^v) \\
&= (g^{d'} \cdot T_1^v, h^{d'} \cdot T_2^v) \\
&= \text{COM}(v, d').
\end{aligned}$$

Following Section 5, if Π_{srCOM} UC-emulates Π_{COM} and Π_{COM} UC-realises $\mathcal{F}_{\text{COM}}$, then Π_{srCOM} UC-realises $\mathcal{F}_{\text{COM}}$. As it was proved in [30] that Π_{COM} UC-realises $\mathcal{F}_{\text{COM}}$, only the former part has to be considered here. It was also shown in Section 5 that if RF in Π_{srCOM} is strongly transparent and anti-signalling, then Π_{srCOM} UC-emulates Π_{COM} in the $\{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{AUTH}}\}$-hybrid model, where the main parties are malicious, semi-honest or subversion corrupted with a specious code $\bar{\pi}$, and the respective sub-parties RF are honest, semi-honest or malicious. It remains to show the transparency and anti-signalling properties.

Lemma 2. *RF in Π_{srCOM} is strongly transparent and anti-signalling.*

Proof. Since the firewall of the verifier only passes through the message, only the firewall of the sender has to be analysed.

The firewall of the sender solely changes the outgoing commitment c to $c' = \text{COM}(0, r) \cdot c = (g^r \cdot T_1^0, h^r \cdot T_2^0) \cdot (g^d \cdot T_1^v, h^d \cdot T_2^v) = (g^{r+d} \cdot T_1^v, h^{r+d} \cdot T_2^v)$. As one can see, this changes the used randomness from d to $r + d$. Considering the transparency definition, d is independent and uniformly random, thus adding an independent uniform random elements r to it does not change this property. Additionally, even if d was output by the random function of the adversary, the sum $r + d$ would still be independent uniform random because of the added randomness by the incorruptible firewall, which is therefore anti-signalling.

The theorem follows.

7 OT

Oblivious transfer is one of the open problems stated by [10]. 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.

While the previously considered commitment scheme was non-interactive and only needed a firewall for the committer, the OT protocol consist 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. Note, that also the subversion code inserted into a sender and a receiver can differ.

7.1 Ideal Functionality: Oblivious Transfer

$\mathcal{F}_{\textbf{OT}}$

Two-party ideal oblivious transfer protocol between a sender S and a receiver $\mathcal{R}$.

- Upon receiving an input $(\texttt{OT-send}, \mathcal{R}, sid, (m_0, m_1))$ from S, where $m_i \in \mathbb{G}$, record the tuple (sid, m_0, m_1) and generate a public delayed output $(\texttt{OT-sent}, S, \mathcal{R}, sid)$ to S. Ignore subsequent commands of the form $(\texttt{OT-send}, \mathcal{R}, sid, \cdot)$.
- Upon receiving an input $(\texttt{OT-receive}, S, sid, b)$ from $\mathcal{R}$, where $b \in \{0, 1\}$, ignore the message if (sid, m_0, m_1) is not recorded. Otherwise, record the tuple (sid, b) and generate a public delayed output$(\texttt{OT-received}, S, sid, m_b)$ to $\mathcal{R}$. Ignore subsequent commands of the form $(\texttt{OT-receive}, S, sid, \cdot)$.

  \\ Corruption Handling:
- On input $(\texttt{secret}, sid, \mathcal{P}, \mathsf{sec})$, store $(\texttt{secret}, sid, \mathcal{P}, \mathsf{sec})$.
  If RF is corrupted send $(\texttt{secret}, sid, \mathcal{P}, \mathsf{sec})$ to Sim.
- Ignore all further inputs.

7.2 Real Protocol: Subversion-resilient Oblivious Transfer based on DLog [30]

We base our protocol on the UC-secure 1-out-of-2 OT protocol given by [30] where one of the messages $m_0, m_1 \in \mathbb{G}$ from the sender S can be obtained by the receiver $\mathcal{R}$. Like the commitment protocol above, it relies on the CRS and in particular, starts very similarly to the commitment. Consequently, as was done for the commitment, the protocol also has to be adapted by making the field element $T_2 \in \mathbb{G}$ part of the CRS. The group $\mathbb{G}$ with generator g and the field $\mathbb{Z}_q$ are public inputs.

The base OT scheme in [30] proves security under malicious corruption. Again, this implies security under augmented semi-honest corruption. Plain semi-honest security, however, has not been shown yet. If one cares or doubt about this, we leave this as an open problem to the reader, as this would have to be proven first for the case above in Theorem 2 where the subverted party and plain semi-honest corrupted firewall behave indistinguishable from a plain semi-honest party.

$$\Pi_{\mathbf{srOT}}^{\mathcal{F}_{\mathbf{CRS}}, \mathcal{F}_{\mathbf{AUTH}}}$$

Realises:
Subversion-resilient two-party oblivious transfer protocol between a main party sender S, its sub-party firewall RF_S, a main party receiver $\mathcal{R}$, its sub-party firewall $RF_{\mathcal{R}}$, and an adversary $\mathcal{A}$.
Parameters:
- Functionality $\mathcal{F}_{\mathrm{AUTH}}$.
- Functionality $\mathcal{F}_{\mathrm{CRS}}$:
  Every party receives on $(\texttt{value}, sid)$ to $\mathcal{F}_{\mathrm{CRS}}$ $(\texttt{crs}, sid, (g, T_1, h, T_2)$.
- Only static subversion corruption is pictured, others omitted.
- Every main-party has an empty initialised secret tape.

Behaviour of Party $\mathcal{R}$

\\ Subversion Corruption Handling:

- Upon receiving ($\texttt{corrupt}, \mathit{sid}, (\texttt{subversion}, \bar{\pi})$) as first message after invocation from $\mathcal{A}$, shell checks if $\bar{\pi}$ is a valid specious code by the criteria given in *Section* 3.2, that is that $\bar{\pi}$ only consists of a randomness function.
  If $\bar{\pi}$ is valid specious, shell replaces all randomness function in the body with the one from $\bar{\pi}$, and sets a subversion corruption flag. Else ignore.
  Ignore subsequent corruption commands ($\texttt{corrupt}, \mathit{sid}, \cdot$).
- Upon receiving ($\texttt{secret}, \mathit{sid}, s$) from $\mathcal{Z}$, write s on the secret tape.
  \\ Oblivious Transfer Handling:
- Upon receiving input ($\texttt{OT-receive}, S, \mathit{sid}, b$) from $\mathcal{Z}$, where $b \in \{0, 1\}$, get α from the randomness function, \\ presumably subverted
  and set $(B, H) := \mathit{Choose}(g, h, T_1, T_2, b, \alpha) = (g^\alpha \cdot T_1^b, h^\alpha \cdot T_2^b)$.
  Send ($\texttt{OT-receive}, S, \mathit{sid}, (B, H)$) to $\mathit{RF}_\mathcal{R}$ via an immediate channel.
- Upon receiving ($\texttt{sent}, S, \mathit{sid}, \mathit{mid}, (\tilde{z}, \tilde{c}_0, \tilde{c}_1)$) from $\mathit{RF}_\mathcal{R}$, set $m_b := \tilde{c}_b \cdot \tilde{z}^{-\alpha}$.
  Output ($\texttt{OT-received}, S, \mathit{sid}, m_b$) to $\mathcal{Z}$.

Behaviour of Sub-Party $\mathit{RF}_\mathcal{R}$

- Upon receiving ($\texttt{OT-receive}, S, \mathit{sid}, (B, H)$) from $\mathcal{R}$, sample $\beta \overset{\$}{\leftarrow} \mathbb{Z}_q$, set $(B', H') := (B \cdot g^\beta, H \cdot h^\beta)$, and send ($\texttt{send}, S, \mathit{sid}, \mathit{mid}, (B', H')$) in the name of $\mathcal{R}$ to $\mathcal{F}_{\text{AUTH}}$.
- Upon receiving ($\texttt{sent}, S, \mathit{sid}, \mathit{mid}, (z', c_0', c_1')$) from $\mathcal{F}_{\text{AUTH}}$, set $(\tilde{z}, \tilde{c}_0, \tilde{c}_0) := (z', c_0' \cdot z'^{-\beta}, c_1' \cdot z'^{-\beta})$, and send ($\texttt{sent}, S, \mathit{sid}, \mathit{mid}, (\tilde{z}, \tilde{c}_0, \tilde{c}_1)$) via and immediate channel to $\mathcal{R}$.

Behaviour of Party S

\\ Subversion Corruption Handling same es for Party $\mathcal{R}$.
\\ Oblivious Transfer Handling:

- Upon receiving ($\texttt{sent}, \mathcal{R}, \mathit{sid}, (B', H')$) from RF_S, output ($\texttt{OT-sent}, \mathcal{R}, \mathit{sid}$) to $\mathcal{Z}$.
- Upon receiving ($\texttt{OT-send}, \mathcal{R}, \mathit{sid}, (m_0, m_1)$) from $\mathcal{Z}$, where $m_0, m_1 \in \mathbb{G}$, sample $r, s \overset{\$}{\leftarrow} \mathbb{Z}_q$, set $(z, c_0, c_1) := \mathit{Transfer}(g, h, T_1, T_2, B', H', m_0, m_1) = (g^r \cdot h^s, B'^r \cdot H'^s \cdot m_0, (B'/T_1)^r \cdot (H'/T_2)^s \cdot m_1)$, and send ($\texttt{OT-send}, \mathcal{R}, \mathit{sid}, (z, c_0, c_1)$) via an immediate channel to RF_S.

Behaviour of Sub-Party RF_S

- Upon receiving ($\texttt{sent}, \mathcal{R}, \mathit{sid}, (B', H')$) from $\mathcal{F}_{\text{AUTH}}$, send ($\texttt{sent}, \mathcal{R}, \mathit{sid}, (B', H')$) via an immediate channel to S.
- Upon receiving ($\texttt{OT-send}, \mathcal{R}, \mathit{sid}, (z, c_0, c_1)$) from S, sample $r', s' \overset{\$}{\leftarrow} \mathbb{Z}_q$, set $(z', c_0', c_1') := (z \cdot g^{r'} \cdot h^{s'}, c_0 \cdot B'^{r'} \cdot H'^{s'}, c_1 \cdot (B'/T_1)^{r'} \cdot (H'/T_2)^{s'})$. and send ($\texttt{send}, \mathcal{R}, \mathit{sid}, \mathit{mid}, (z', c_0', c_1')$) in the name of S to $\mathcal{F}_{\text{AUTH}}$.

Behaviour of Party $\mathcal{A}$:

\\ Subversion Corruption Handling:

- Upon receiving input ($\texttt{corrupt}, \mathit{sid}, \mathcal{P}, (\texttt{subversion}, \bar{\pi})$) from $\mathcal{Z}$, send ($\texttt{corrupt}, \mathit{sid}, (\texttt{subversion}, \bar{\pi})$) to backdoor tape of $\mathcal{P}$.
  \\ AUTH Handling:
- Upon receiving input ($\texttt{send}, \mathit{sid}, \mathcal{P}_1, \mathcal{P}_2, \mathit{msg}$) from $\mathcal{F}_{\text{AUTH}}$, output ($\texttt{send}, \mathit{sid}, \mathcal{P}_1, \mathcal{P}_2, \mathit{msg}$) to $\mathcal{Z}$.
- Upon receiving input ($\texttt{sendok}, \mathit{sid}$) from $\mathcal{Z}$, forward ($\texttt{sendok}, \mathit{sid}$) to $\mathcal{F}_{\text{AUTH}}$.

7.3 Security Proof

Theorem 4. *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 malicious, semi-honest or subversion corrupted with a specious code $\bar{\pi}$, and the respective sub-parties RF are honest, semi-honest or malicious.*

Proof. Since the firewall only rerandomises the used randomness, one can easily see that Π_{srOT} is correct;

Lemma 3. *Π_{srOT} is correct.*

Proof. Π_{srOT} is correct if the receiver $\mathcal{R}$ can open the chosen message m_b, that is $m_b = \tilde{c}_b \cdot \tilde{z}^{-\alpha}$.

We consider the cases separately:

– Case $b = 0$:

$$\begin{aligned}\tilde{c}_0 \cdot \tilde{z}^{-\alpha} &= c'_0 \cdot z'^{-\beta} \cdot z'^{-\alpha}\\ &= B'^{r+r'} \cdot H'^{s+s'} \cdot m_0 \cdot g^{-(\alpha+\beta)(r+r')} \cdot h^{-(\alpha+\beta)(s+s')}\\ &= g^{(\alpha+\beta)\cdot(r+r')} \cdot h^{(\alpha+\beta)\cdot(s+s')} \cdot m_0 \cdot g^{-(\alpha+\beta)(r+r')} \cdot h^{-(\alpha+\beta)(s+s')}\\ &= m_0.\end{aligned}$$

– Case $b = 1$:

$$\begin{aligned}\tilde{c}_1 \cdot \tilde{z}^{-\alpha} &= c'_1 \cdot z'^{-\beta} \cdot z'^{-\alpha}\\ &= (B'/T_1)^{r+r'} \cdot (H'/T_2)^{s+s'} \cdot m_1 \cdot g^{-(\alpha+\beta)(r+r')} \cdot h^{-(\alpha+\beta)(s+s')}\\ &= g^{(\alpha+\beta)\cdot(r+r')} \cdot h^{(\alpha+\beta)\cdot(s+s')} \cdot m_1 \cdot g^{-(\alpha+\beta)(r+r')} \cdot h^{-(\alpha+\beta)(s+s')}\\ &= m_1.\end{aligned}$$

Following Section 5, if Π_{srOT} UC-emulates Π_{OT} and Π_{OT} UC-realises $\mathcal{F}_{\mathrm{OT}}$, then Π_{srOT} UC-realises $\mathcal{F}_{\mathrm{OT}}$. As it was proved in [30] that Π_{OT} UC-realises $\mathcal{F}_{\mathrm{OT}}$, only the former part has to be considered here. It was also shown in Section 5 that if *RF* in Π_{srOT} is strongly transparent and anti-signalling, then Π_{srOT} UC-emulates Π_{OT} in the $\{\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{AUTH}}\}$-hybrid model, where the main parties are malicious, semi-honest or subversion corrupted with a specious code $\bar{\pi}$, and the respective sub-parties *RF* are honest, semi-honest or malicious. It remains to show the transparency and anti-signalling properties.

Lemma 4. *RF in Π_{srOT} is strongly transparent and anti-signalling.*

Proof. We consider the firewalls for the sender and the receiver separately.

Sender The firewall of the sender only changes the outgoing tuple (z, c_0, c_1) to

$$\begin{aligned}(z', c'_0, c'_1) &= (z \cdot g^{r'} \cdot h^{s'}, c_0 \cdot B'^{r'} \cdot H'^{s'}, c_1 \cdot (B'/T_1)^{r'} \cdot (H'/T_2)^{s'})\\ &= (g^{r+r'} \cdot h^{s+s'}, B'^{r+r'} \cdot H'^{s+s'} \cdot m_0, (B'/T_1)^{r+r'} \cdot (H'/T_2)^{s+s'} \cdot m_1).\end{aligned}$$

As one can see, this changes the used randomness from r and s to $r + r'$ and $s + s'$. Considering the transparency definition, r and s are independent and uniformly random, thus adding independent uniform random elements r', s' to it does not change this property. Additionally, even if r, s were output by the random function of the adversary, the sums $r + r'$ and $s + s'$ would still be independent uniform random because of the added randomness by the incorruptible firewall, which is therefore anti-signalling.

Receiver This firewall changes both the tuple (B, H) output by the receiver to $(B', H') = (B \cdot g^{\beta}, H \cdot h^{\beta}) = (g^{\alpha+\beta} \cdot T_1^b, h^{\alpha+\beta} \cdot T_2^b)$ and (z', c_0', c_0') to

$$\begin{aligned}(\tilde{z}, \tilde{c}_0, \tilde{c}_0) &= (z', c_0' \cdot z'^{-\beta}, c_1' \cdot z'^{-\beta}) \\ &= (g^{r+r'} \cdot h^{s+s'}, B^{r+r'} \cdot H^{s+s'} \cdot m_0, (B/T_1)^{r+r'} \cdot (H/T_2)^{s+s'} \cdot m_1).\end{aligned}$$

Again, one can see that this only changes the used randomness from α to $\alpha + \beta$, which is reverted back before the message from the sender is passed to the receiver. As this implies that the second has the same distribution as in the underlying protocol, only the first message has to further be analysed. Considering the transparency definition, α is independent and uniformly random as is β, hence their sum is as well. Additionally, even if α was output by the random function of the adversary, the sum $\alpha + \beta$ would still be independent uniform random because of the added randomness by the incorruptible firewall, which is therefore anti-signalling.

The theorem follows.

8 Circumvention of impersonation attack

In the previous sections, we showed that an honest firewall can indeed remove the leakage given by a subverted implementation completely. Hence, the security level of the system is improved in this case. While our analysis shows that all corruption cases are simulatable, a malicious firewall can now impersonate the combined party, even if Alice is honest herself. This is consistent with the handling due to Chakraborty et al. [10]. The attentive reader might now ask why they should use such a firewall as the security in one scenario improves, while the security in another scenario declines drastically. In this section, we will first present a careful analysis that will show that the only damage which could be caused by such a corrupted firewall lies in the danger of *impersonation attacks*. Then, we will discuss different approaches to handle this situation. In the first approach, we will add an ideal functionality $\mathcal{F}_{\text{chAUTH}}$, similar to the sanitised authenticated channels used by Chakraborty et al. This functionality will exactly cover the possibilities given to a corrupted firewall. We describe how to add such a functionality to our commitment protocol to protect against all impersonation attacks. Afterwards, we show two realisations of these functionality: A very efficient one for our commitment protocol based on signatures and a more general version using zero-knowledge proofs for our oblivious transfer protocol.

The general version is also applicable to a wide range of other protocols. We note that in [10], the employment of a three-tier model is discussed where a third component is added to the party and firewall. This could also be used in our model. Finally, we discuss that many security guarantees might still be guaranteed even in the presence of an attacker with impersonation capabilities. We note here that all adaptions or extensions to the base protocols presented here mean that we lose the transitivity of the UC emulation. Hence, we either need to directly show the simulation in the ideal world or need to take a two-step approach.

8.1 Impersonation Attacks

In this section, we first want to understand the capabilities of a corrupted firewall. The importance of this was already mentioned explicitly by Dodis et al.

> [15] *Second, and more importantly, this definitional choice provides an elegant solution to a natural concern about reverse firewalls: What happens when the firewall itself is corrupted? Of course, if both Alices own machine and her firewall are compromised, then we cannot possibly hope for security. But, if Alices own implementation is correct and the firewall has been corrupted, then we can view the firewall as "part of" the adversary in the firewall-free protocol between Alice and Bob. Since this underlying protocol must itself be secure, it trivially remains secure in the presence of a corrupted firewall.*

As indicated by the above quote, the problem could very easily be handled in papers using game-based approaches by treating the corrupted firewall as part of the other party. Our notion of transparency also allows us to treat the combination of the firewall and the *outside* parties as a maliciously corrupted party and the usual security guarantees against such parties thus imply security. Hence, even if using a maliciously corrupted firewall which could isolate the party from the other one, no security would be lost. Note that this argument heavily requires transparency of the firewall. In contrast, Chakraborty et al. who also introduce the corruption case of *isolation*, are not able to make use of the security guarantees of the underlying protocol in a black-box manner, as their solution has a feedback channel and the party thus needs to communicate explicitly with the firewall. Hence, as shown in the following quote, they are not able to provide any security guarantee in this scenario.

> [10] *The Isolated case corresponds to the situation where the core is honest and the firewall is corrupted, and thus the firewall is isolating the core from the network. This will typically correspond to a corrupted party. However, in some cases, some partial security might be obtainable, like the inputs of the core being kept secret.*

As described above, the notion of strong transparency intuitively allows to guarantee that Alice's secret is not leaked even if the firewall is maliciously corrupted by treating the firewall as part of a maliciously corrupted Bob. While this argument is true in the context of the protocols studied by Mironov and Stephens-Davidowitz [14] and Dodic, Mironov and Stephens-Davidowitz [15], it does *not* hold in general. Careful consideration shows that the protocols studied in the previous works did not require any form of *authentication*. However, as the complete communication between the parties needs to be modeled in the UC framework, questions about authentication need to be dealt with (typically by using the ideal functionality $\mathcal{F}_{\text{AUTH}}$). Hence, when studying the capabilities of a corrupted firewall in the UC model, we need to consider the following four points:

1. One can easily see that we cannot guarantee that the system is non-signalling as the firewall can just pass the output of the party through (in case of a subverted party).
2. Additionally, the firewall can either isolate the party from the protocol or send messages in their name.
3. Hence, when running a priorly given protocol, the firewall could impersonate its party.
4. But can we still provide some guarantees of the security of the underlying protocol?

To make this discussion more explicit, let us take a look at the example of a commitment scheme:

In the case of the commitment scheme (presented in Section 6), from the viewpoint of the other parties, a maliciously corrupted RF ...

1. ... could pass through the output from Alice, thus revealing the leakage if subverted.
2. ... could rerandomise the output from Alice.
3. ... could isolate Alice from the protocol by not sending anything, though this would probably soon be detected by Bob or a higher protocol.
4. ... could impersonate Alice by sending a new commitment to a new chosen message.

In the following discussion, we will ignore the third case, as a lack of communication will usually not invalid any security guarantees. We will now discuss the different interesting scenarios and their implications with regard to the corruption status of the combined party.

If Alice is honest, then the first case would guarantee a completely honest combined party. Similarly, the combined party would also behave honestly in the second case, as the malicious firewall could only rerandomise the uniformly generated randomness, which would still lead to uniformly generated randomness. However, in the fourth case, the corrupted firewall (and thus the attacker) can send messages in the name of Alice, which is clearly a non-honest behaviour and the combined party thus needs be treated as corrupted.

If Alice is subverted, the first two cases would lead to a semi-honest combined party, as the attacker would learn the secret which could potentially contain the complete state of Alice. Furthermore, in the fourth case, we again encounter the situation that the combined party needs to be treated as maliciously corrupted.

Hence, in order to prevent the security problems due to a corrupted firewall, it is sufficient to solve the before mentioned impersonation problem. We note here that a dedicated lower corruption setting of impersonation could also be of help here instead of the over-approximation of a maliciously corrupted combined party.

8.2 Adding $\mathcal{F}_{\mathrm{chAUTH}}$ to Π_{srCOM}

In this section we show how to handle the impersonation problem by considering a new ideal functionality $\mathcal{F}_{\mathrm{chAUTH}}$. The main idea behind this functionality is to handle the fourth case described above where the maliciously corrupted firewall does not perform a rerandomisation (even with bad randomness), but sends a completely different message. Hence, $\mathcal{F}_{\mathrm{chAUTH}}$ will guarantee that the output of the firewall is a rerandomised version of the output of Alice. More generally, we assume that if Alice outputs x, then an honest firewall would output a value y such that $(x, y) \in R$ for some NP-relation R. We denote a witness proving that $(x, y) \in R$ by w and assume that the honest firewall produces such a witness when generating y. For the sake of simplicity, we directly combine this functionality with our authenticated channels but only explain the changes due to the check here. After the functionality sends the backdoor message $(\mathtt{send}, sid, \mathcal{P}_1, \mathcal{P}_2, y)$ to $\mathcal{A}$, it continues as in $\mathcal{F}_{\mathrm{AUTH}}$.

$\mathcal{F}_{\mathbf{chAUTH}R}$

Provides:
Authenticated channel between $\mathcal{P}_1$ and $\mathcal{P}_2$ that checks whether the output y of the firewall of $\mathcal{P}_1$ and the output x of $\mathcal{P}_1$ fulfil $(x, y) \in R$.
Behaviour:

- Interface IO
  - Upon invocation with input $(\mathtt{check}, sid, x)$ from $\mathcal{P}_1$, record the tuple (sid, x) and return $(\mathtt{checked}, sid, \mathtt{OK})$ to $\mathcal{P}_1$. Ignore subsequent commands of the form $(\mathtt{check}, sid, x)$.
  - On input $(\mathtt{send}, sid, (y, w))$ from RF, ignore the message if $(\mathtt{check}, sid, x)$ is not recorded. If $(x, y) \notin R$ (using w), ignore the message. If $(x, y) \in R$ (using w), send backdoor message $(\mathtt{send}, sid, \mathcal{P}_1, \mathcal{P}_2, y)$ to $\mathcal{A}$. Ignore subsequent commands of the form $(\mathtt{send}, sid, (y, w))$.

In the concrete case of our commitment protocol, x would correspond to a commitment c and the relation R corresponds to all y that are rerandomisations of c, i.e.,

$$R = \{(c, c') \mid \exists r' \in \{0,1\}^* : c' = \mathrm{COM}(0, r) \cdot c\}.$$

Finally, the witness w is simply the random string r' used by the firewall to rererandomise c.

In the following, we show how to construct a simulator handling the case that the party Alice is honest and the firewall is maliciously corrupted in the presence of $\mathcal{F}_{\text{chAUTH}}$. This simulator shows that we can treat this case as though as Alice is honest and the firewall is not present.

Execution in the Real World: To commit a value, the execution in the real world is as follows: First, the environment $\mathcal{Z}$ sends some value v to Alice who then computes a corresponding commitment c using randomness r. This commitment c is then given to $\mathcal{F}_{\text{chAUTH}}$ that returns `OK` to Alice. Now, Alice send c to the maliciously corrupted firewall who forwards it to the environment $\mathcal{Z}$. The environment now sends some commitment c' and the randomness r' to $\mathcal{F}_{\text{chAUTH}}$ (in the name of the dummy attacker). The ideal functionality $\mathcal{F}_{\text{chAUTH}}$ now checks whether $c = \text{COM}(0, r') \cdot c$ and returns c' to the attacker (resp. the environment). Now, if $\mathcal{Z}$, sends `OK`, $\mathcal{F}_{\text{chAUTH}}$ sends c' to the second party Bob.

To open a value, the execution in the real world is as follows: First, the environment $\mathcal{Z}$ sends `open` to Alice who then sends the pair (v, r) to $\mathcal{F}_{\text{chAUTH}}$ and is given `OK`. Now, Alice sends (v, r) to the maliciously corrupted firewall, who forwards this pair to the environment (via the dummy attacker). The environment $\mathcal{Z}$ now sends a pair $(\bar{v}, \bar{r})$ to $\mathcal{F}_{\text{chAUTH}}$ who returns it directly to the environment. After $\mathcal{Z}$ sends `OK`, the functionality $\mathcal{F}_{\text{chAUTH}}$ sends $(\bar{v}, \bar{r})$ to Bob.

Simulation in the Ideal World: To handle the commitment, our simulator *Sim* works as follows: First, $\mathcal{Z}$ sends v to Alice, who then calculates the corresponding commitment c using randomness r. This commitment c is then given to $\mathcal{F}_{\text{chAUTH}}$ that returns `OK` to Alice. Now, Alice send c to the adversary, which is the simulator *Sim*. The simulator now sends c in the name of the malicious firewall to the dummy attack which outputs it to $\mathcal{Z}$. The environment now sends a commitment c' and the randomness r' in the name of the firewall to *Sim*. The simulator now checks whether $c' = \text{COM}(0, r') \cdot c$ and sends c' to the dummy attacker, who forwards it to $\mathcal{Z}$. If $c' \neq \text{COM}(0, r') \cdot c$, the simulator ignores the message. If the environment $\mathcal{Z}$ sends `OK`, the simulator forwards c' to Bob.

To handle the opening, our simulator *Sim* works as follows: First, $\mathcal{Z}$ sends `open` to Alice who then sends (v, r) to $\mathcal{F}_{\text{chAUTH}}$. The pair (v, r) is then forwarded to the attacker, which is *Sim*. The simulator sends (v, r) to the environment (using the dummy attacker controlling the corrupted firewall). The environment then sends $(\bar{v}, \bar{r})$ to the simulator, which forwards it to the dummy attacker and then to the environment. If the environment $\mathcal{Z}$ sends `OK`, the simulator forwards $(\bar{v}, \bar{r})$ to Bob.

8.3 Circumventing Impersonation on Protocol Side

As shown above, the ideal functionality $\mathcal{F}_{\text{chAUTH}}$ can be used to prevent impersonation attacks. In this section, we will show two possible ways how to concretely realise these ideal functionality exemplarily for both commitments (Section 6) and oblivious transfer (Section 7). Our solution for commitments will be very efficient, but specifically tailored to commitments. In contrast, our solution for oblivious transfer is not as efficient, but very general and should work for all protocols.

Protecting commitments through signatures While we constructed bit commitments in Section 6, we will now consider string commitments. We note that the composition theorem simply allows us to build string commitments out of bit commitments by combining the single bit commitments. In the following, we will now make use of *unique signatures*, i.e., a signature scheme where each message has exactly one valid signature. See, e.g., [52, 53] for more information and constructions of such signatures. We assume that Alice hold some signing key sk and Bob holds the verification key vk which was distributed, e.g., via a PKI. Now, if Alice wants to commit to a value v, we first compute such unique signature, denoted as $\sigma(v)$. Since the signatures are unique, they contain no randomness and, therefore, cannot any leak information about the secret. Then, we compute the string commitment c' of the value $v' = v \| \sigma(v)$. Once the commitment is opened, Bob will perform an additional verification step using $\mathtt{Vfy}$ to check whether the signature $\sigma(v)$ contained in the commitment $c' = \mathrm{COM}(v \| \sigma(v))$ is a valid signature of the value v. Now, if the adversary sends a commitment which passes the verification check of Bob, we know that it is a rerandomisation of c'. Otherwise, c' would contain a forgery which would contradict the security of the signature scheme.

Protecting oblivious transfer through zero-knowledge proofs Oblivious transfer is a slightly more difficult protocol than commitments, as here neither the information of which message was chosen b nor the message not chosen m_{1-b} will be revealed which makes our above solution not directly transferable. Nevertheless, we will also make use of signature schemes, which here do not need to be unique. We again assume that that Alice hold some signing key sk and Bob holds the verification key vk which was distributed, e.g., via a PKI. Now, whenever a party sends the value x, it will also send a signature σ of x along with it. When the firewall is given x and σ, it first produces the output y along with the witness w for $(x, y) \in R$ along with a zero-knowledge proof ρ of the language

$$\mathcal{L} = \{(x, \sigma, w) \mid (x, y) \in R \text{ via } w \land \mathtt{Vfy}(vk, x, \sigma) = 1\}.$$

The firewall now sends y along with ρ. When obtaining y and ρ, the receiving party can verify that y is a valid rerandomisation by checking ρ. Due to the zero-knowledge property of ρ, the receiving party does not learn anything about the (possibly leaking) values x and σ. The soundness property of ρ guarantees that ρ contains a valid proof, and thus in turn a valid value-signature pair (x, σ). To produce a valid proof for y, the firewall can thus only rerandomise x.

8.4 Remaining Security Guarantees in the presence of Impersonation

As shown above, there are solutions to avoid the impersonation problems and thus we can avoid to treat the combined party as maliciously corrupted. However, it comes with a cost as we need to modify the underlying protocol when realising this functionality. Without any changes to the protocol, it seems unlikely that we can handle the impersonation problem. However, some security guarantees might still be maintained, hence providing a graceful degradation. This can, for example, be of interest to Alice if she primarily wants to secure her committed message.

Let's again consider the situation of a commitment here: Since the rerandomisation check was not added to $\mathcal{F}_{\mathrm{AUTH}}$ itself, a simulation is no longer possible: The simulator does not know whether the output c' of a firewall is a rerandomisation of the original commitment c or a completely new commitment due to the hiding property of the commitment scheme. Nevertheless, a careful analysis shows that both the hiding and binding of Alice's commitment still hold in this case, as the malicious firewall is neither able to look into the original commitment c nor to open it to a different value. The only possible attack is that the value given to Bob is either wrong or that Bob does not receive any message. This also extends to other functionalities like the coin toss which is still efficiently possible.

It is interesting to see that we cannot capture all security guarantees by UC-simulation. The impersonation corruption might be interesting for further research. But the loss of Alice of hiding of the commitment if we treat the composed party as malicious is not expected in the case where Alice is honest but her firewall is malicious. We could also circumvent the impersonation by giving the simulator more power, i.e. that it can impersonate still in the opening phase. But this would imply that the ideal commitment is equivocable.

If the composed party behaviour shall exactly represent the natural security guaranties of the non-composed setting, one has to adapt the UC-model in extreme ways and possible still not all security guaranties are simulated, which implies limits of the (UC)-simulation as gold standard for security analysis.

References

1. Simmons, G.J.: The history of subliminal channels. IEEE J. Sel. Areas Commun. **16**(4), 452–462 (1998)
2. Young, A., Yung, M.: Kleptography: Using Cryptography Against Cryptography. In: Fumy, W. (ed.) Advances in Cryptology – EUROCRYPT'97. Lecture Notes in Computer Science, pp. 62–74. Springer, Heidelberg, Germany, Konstanz, Germany (1997). https://doi.org/10.1007/3-540-69053-0_6
3. Young, A., Yung, M.: The Dark Side of "Black-Box" Cryptography, or: Should We Trust Capstone? In: Koblitz, N. (ed.) Advances in Cryptology – CRYPTO'96. Lecture Notes in Computer Science, pp. 89–103. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (1996). https://doi.org/10.1007/3-540-68697-5_8
4. 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: Fu, K., Jung, J. (eds.) USENIX Security 2014: 23rd USENIX Security Symposium, pp. 319–335. USENIX Association, San Diego, CA, USA (2014)
5. Ball, J., Borger, J., Greenwald, G., *et al.*: Revealed: how US and UK spy agencies defeat internet privacy and security, The Guardian (6 September 2013).

6. Bellare, M., Paterson, K.G., Rogaway, P.: Security of Symmetric Encryption against Mass Surveillance. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014, Part I. Lecture Notes in Computer Science, pp. 1–19. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (2014). https://doi.org/10.1007/978-3-662-44371-2_1
7. Bellare, M., Jaeger, J., Kane, D.: Mass-surveillance without the State: Strongly Undetectable Algorithm-Substitution Attacks. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015: 22nd Conference on Computer and Communications Security, pp. 1431–1440. ACM Press, Denver, CO, USA (2015). https://doi.org/10.1145/2810103.2813681
8. 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.
9. Berndt, S., Liskiewicz, M.: Algorithm Substitution Attacks from a Steganographic Perspective. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th Conference on Computer and Communications Security, pp. 1649–1660. ACM Press, Dallas, TX, USA (2017). https://doi.org/10.1145/3133956.3133981
10. Chakraborty, S., Magri, B., Nielsen, J.B., Venturi, D.: Universally Composable Subversion-Resilient Cryptography. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part I. Lecture Notes in Computer Science, pp. 272–302. Springer, Heidelberg, Germany, Trondheim, Norway (2022). https://doi.org/10.1007/978-3-031-06944-4_10
11. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, pp. 136–145 (2001). https://doi.org/10.1109/SFCS.2001.959888
12. Canetti, R.: Universally Composable Security. J. ACM **67**(5), 28:1–28:94 (2020)
13. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In: Aho, A. (ed.) 19th Annual ACM Symposium on Theory of Computing, pp. 218–229. ACM Press, New York City, NY, USA (1987). https://doi.org/10.1145/28395.28420
14. Mironov, I., Stephens-Davidowitz, N.: Cryptographic Reverse Firewalls. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part II. Lecture Notes in Computer Science, pp. 657–686. Springer, Heidelberg, Germany, Sofia, Bulgaria (2015). https://doi.org/10.1007/978-3-662-46803-6_22
15. Dodis, Y., Mironov, I., Stephens-Davidowitz, N.: Message Transmission with Reverse Firewalls—Secure Communication on Corrupted Machines. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016, Part I. Lecture Notes in Computer Science, pp. 341–372. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (2016). https://doi.org/10.1007/978-3-662-53018-4_13
16. Chakraborty, S., Ganesh, C., Sarkar, P.: Reverse Firewalls for Oblivious Transfer Extension and Applications to Zero-Knowledge. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part I. Lecture Notes in Computer Science, pp. 239–270. Springer, Heidelberg, Germany, Lyon, France (2023). https://doi.org/10.1007/978-3-031-30545-0_9
17. Chakraborty, S., Magliocco, L., Magri, B., Venturi, D.: Key Exchange in the Post-Snowden Era: UC Secure Subversion-Resilient PAKE. IACR Cryptol. ePrint Arch. (2023)
18. Chen, R., Huang, X., Yung, M.: Subvert KEM to Break DEM: Practical Algorithm-Substitution Attacks on Public-Key Encryption. In: Moriai, S., Wang, H. (eds.)

Advances in Cryptology – ASIACRYPT 2020, Part II. Lecture Notes in Computer Science, pp. 98–128. Springer, Heidelberg, Germany, Daejeon, South Korea (2020). https://doi.org/10.1007/978-3-030-64834-3_4

19. Berndt, S., Wichelmann, J., Pott, C., Traving, T.-H., Eisenbarth, T.: ASAP: Algorithm Substitution Attacks on Cryptographic Protocols. In: Suga, Y., Sakurai, K., Ding, X., Sako, K. (eds.) ASIACCS 22: 17th ACM Symposium on Information, Computer and Communications Security, pp. 712–726. ACM Press, Nagasaki, Japan (2022). https://doi.org/10.1145/3488932.3517387
20. 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)
21. Degabriele, J.P., Farshim, P., Poettering, B.: A More Cautious Approach to Security Against Mass Surveillance. In: Leander, G. (ed.) Fast Software Encryption – FSE 2015. Lecture Notes in Computer Science, pp. 579–598. Springer, Heidelberg, Germany, Istanbul, Turkey (2015). https://doi.org/10.1007/978-3-662-48116-5_28
22. Ateniese, G., Magri, B., Venturi, D.: Subversion-Resilient Signature Schemes. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015: 22nd Conference on Computer and Communications Security, pp. 364–375. ACM Press, Denver, CO, USA (2015). https://doi.org/10.1145/2810103.2813635
23. Armour, M., Poettering, B.: Algorithm substitution attacks against receivers. Int. J. Inf. Sec. **21**(5), 1027–1050 (2022)
24. Teseleanu, G.: Unifying Kleptographic Attacks. In: NordSec. Lecture Notes in Computer Science, pp. 73–87. Springer (2018)
25. 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)
26. Young, A., Yung, M.: The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems. In: Kaliski Jr., B.S. (ed.) Advances in Cryptology – CRYPTO'97. Lecture Notes in Computer Science, pp. 264–276. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (1997). https://doi.org/10.1007/BFb0052241
27. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th Annual ACM Symposium on Theory of Computing, pp. 21–30. ACM Press, San Diego, CA, USA (2007). https://doi.org/10.1145/1250790.1250794
28. Russell, A., Tang, Q., Yung, M., Zhou, H.-S.: Cliptography: Clipping the Power of Kleptographic Attacks. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016, Part II. Lecture Notes in Computer Science, pp. 34–64. Springer, Heidelberg, Germany, Hanoi, Vietnam (2016). https://doi.org/10.1007/978-3-662-53890-6_2
29. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. IACR Cryptol. ePrint Arch. (2000)
30. Canetti, R., Sarkar, P., Wang, X.: Efficient and Round-Optimal Oblivious Transfer and Commitment with Adaptive Security. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020, Part III. Lecture Notes in Computer Science, pp. 277–308. Springer, Heidelberg, Germany, Daejeon, South Korea (2020). https://doi.org/10.1007/978-3-030-64840-4_10
31. Ganesh, C., Magri, B., Venturi, D.: Cryptographic Reverse Firewalls for Interactive Proof Systems. In: Czumaj, A., Dawar, A., Merelli, E. (eds.) ICALP 2020: 47th International Colloquium on Automata, Languages and Programming. LIPIcs, 55:1–

55:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Saarbrücken, Germany (2020). https://doi.org/10.4230/LIPIcs.ICALP.2020.55

32. Bossuat, A., Bultel, X., Fouque, P.-A., Onete, C., van der Merwe, T.: Designing Reverse Firewalls for the Real World. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) ESORICS 2020: 25th European Symposium on Research in Computer Security, Part I. Lecture Notes in Computer Science, pp. 193–213. Springer, Heidelberg, Germany, Guildford, UK (2020). https://doi.org/10.1007/978-3-030-58951-6_10
33. Chakraborty, S., Dziembowski, S., Nielsen, J.B.: Reverse Firewalls for Actively Secure MPCs. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part II. Lecture Notes in Computer Science, pp. 732–762. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (2020). https://doi.org/10.1007/978-3-030-56880-1_26
34. Chakraborty, S., Ganesh, C., Pancholi, M., Sarkar, P.: Reverse Firewalls for Adaptively Secure MPC Without Setup. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021, Part II. Lecture Notes in Computer Science, pp. 335–364. Springer, Heidelberg, Germany, Singapore (2021). https://doi.org/10.1007/978-3-030-92075-3_12
35. Alwen, J., shelat, a., Visconti, I.: Collusion-Free Protocols in the Mediated Model. In: Wagner, D. (ed.) Advances in Cryptology – CRYPTO 2008. Lecture Notes in Computer Science, pp. 497–514. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (2008). https://doi.org/10.1007/978-3-540-85174-5_28
36. Lepinski, M., Micali, S., shelat, a.: Collusion-free protocols. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing, pp. 543–552. ACM Press, Baltimore, MA, USA (2005). https://doi.org/10.1145/1060590.1060671
37. 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)
38. Bemmann, P., Chen, R., Jager, T.: Subversion-Resilient Public Key Encryption with Practical Watchdogs. In: Garay, J. (ed.) PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, pp. 627–658. Springer, Heidelberg, Germany, Virtual Event (2021). https://doi.org/10.1007/978-3-030-75245-3_23
39. Russell, A., Tang, Q., Yung, M., Zhou, H.-S.: Generic Semantic Security against a Kleptographic Adversary. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th Conference on Computer and Communications Security, pp. 907–922. ACM Press, Dallas, TX, USA (2017). https://doi.org/10.1145/3133956.3133993
40. Bemmann, P., Berndt, S., Diemert, D., Eisenbarth, T., Jager, T.: Subversion-Resilient Authenticated Encryption Without Random Oracles. In: ACNS. Lecture Notes in Computer Science, pp. 460–483. Springer (2023)
41. Russell, A., Tang, Q., Yung, M., Zhou, H.-S.: Correcting Subverted Random Oracles. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, pp. 241–271. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (2018). https://doi.org/10.1007/978-3-319-96881-0_9
42. Chow, S.S.M., Russell, A., Tang, Q., Yung, M., Zhao, Y., Zhou, H.-S.: Let a Non-barking Watchdog Bite: Cliptographic Signatures with an Offline Watchdog. In: Lin, D., Sako, K. (eds.) PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer

Science, pp. 221–251. Springer, Heidelberg, Germany, Beijing, China (2019). https://doi.org/10.1007/978-3-030-17253-4_8

43. Bemmann, P., Berndt, S., Chen, R.: Subversion-Resilient Signatures Without Random Oracles. In: ACNS. Lecture Notes in Computer Science, (in print). Springer (2024)
44. Fischlin, M., Mazaheri, S.: Self-Guarding Cryptographic Protocols against Algorithm Substitution Attacks. In: Chong, S., Delaune, S. (eds.) CSF 2018: IEEE 31st Computer Security Foundations Symposium, pp. 76–90. IEEE Computer Society Press, Oxford, UK (2018). https://doi.org/10.1109/CSF.2018.00013
45. Abdolmaleki, B., Fleischhacker, N., Goyal, V., Jain, A., Malavolta, G.: Steganography-Free Zero-Knowledge. In: TCC (1). Lecture Notes in Computer Science, pp. 143–172. Springer (2022)
46. Badertscher, C., Ciampi, M., Kiayias, A.: Agile Cryptography: A Universally Composable Approach. In: TCC. Lecture Notes in Computer Science, (in print). Springer (2023)
47. Canetti, R., Fischlin, M.: Universally Composable Commitments. In: Kilian, J. (ed.) Advances in Cryptology – CRYPTO 2001. Lecture Notes in Computer Science, pp. 19–40. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (2001). https://doi.org/10.1007/3-540-44647-8_2
48. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols, Cryptology ePrint Archive, Paper 2000/067 (2000). https://eprint.iacr.org/2000/067. https://eprint.iacr.org/2000/067.
49. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: 42nd Annual Symposium on Foundations of Computer Science, pp. 136–145. IEEE Computer Society Press, Las Vegas, NV, USA (2001). https://doi.org/10.1109/SFCS.2001.959888
50. Canetti, R., Hogan, K., Malhotra, A., Varia, M.: A Universally Composable Treatment of Network Time. In: Köpf, B., Chong, S. (eds.) CSF 2017: IEEE 30th Computer Security Foundations Symposium, pp. 360–375. IEEE Computer Society Press, Santa Barbara, CA, USA (2017). https://doi.org/10.1109/CSF.2017.38
51. Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols - Techniques and Constructions. Springer (2010)
52. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable Random Functions. In: 40th Annual Symposium on Foundations of Computer Science, pp. 120–130. IEEE Computer Society Press, New York, NY, USA (1999). https://doi.org/10.1109/SFFCS.1999.814584
53. Hofheinz, D., Jager, T.: Verifiable Random Functions from Standard Assumptions, Cryptology ePrint Archive, Report 2015/1048 (2015). https://eprint.iacr.org/2015/1048.