

Impersonation Resilience of Subversion-Resilient UC Protocols^{*}

Paula Arnold¹, Sebastian Berndt², Jörn Müller-Quade^{3,4}, Astrid Ottenhues^{3,4}, and Johannes Ottenhues^{3,4}

¹ University of Luebeck

`p.arnold@uni-luebeck.de`

² Technische Hochschule Lübeck

`sebastian.berndt@th-luebeck.de`

³ Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

⁴ KASTEL Security Research Labs, Karlsruhe, Germany

`{mueller-quade, astrid.ottenhues, johannes.ottenhues}@kit.edu`

Abstract. Subversion attacks where the attacker aims to replace parts of a cryptographic system by manipulated parts in an undetectable manner have been shown to model both theoretical and practical attacks such as supply chain attacks. Originally proposed by Young and Yung (CRYPTO 1996), interest in them has increased dramatically after the revelations about the inner workings of intelligence agencies due to Snowden and the following paper by Bellare, Paterson, and Rogaway (CRYPTO 2014).

One of the most promising countermeasures against such attacks are cryptographic reverse firewalls, proposed by Mironov and Stephens-Davidowitz (EUROCRYPT 2015), which are small devices used by the parties with the aim to rerandomize the protocol message to remove possible leakage. A wide range of protocols and corresponding firewalls have been developed based on this notion. The usage of such firewalls, however, also introduces another attack surface as an attacker might also corrupt these firewalls. In most works, this problem is either not addressed or treated as if in this case, the complete party using the firewall is compromised. However, as firewalls typically only perform rerandomization without knowledge of secret key material, this is a very coarse over-approximation, as an honest party Alice is now treated as maliciously corrupted. Instead, the main security problem comes from the fact that the firewall can *impersonate* Alice.

In this paper, we introduce the notion of impersonation resilience, which allows us to develop protocols that are also secure against such a corrupted firewall and thus result in an honest complete party. We show a generic compiler that transforms subversion-resilient protocols for commitment or coin toss into protocols that are also impersonation resilient. Following a recent research line, our protocols and security analysis are built on the Universal Composability (UC) framework of Canetti. Of independent interest, we show that existing UC subversion models are either too strong and do not allow secure string commitments, or too weak and allow trivially secure protocols. We addressed this by developing a new, intermediate UC subversion model.

Keywords: Subversion Resilience · Universal Composability · Reverse Firewalls · Impersonation · Security Model

^{*} This article is based on an earlier article: P. Arnold, S. Berndt, J. Müller-Quade, A. Ottenhues, J. Ottenhues: Impersonation Resilience of Subversion-Resilient UC Protocols, ASIACRYPT 2026, © IACR 2026, DOI: <to appear>.

1 Introduction

Modern cryptography with its ability to construct secure protocols relying on the hardness of certain well defined problems has shown to be an incredibly successful approach. This “provable security” approach has been so successful that a large number of modern attacks do not directly attack the underlying problem, but rather target implementation aspects. One such aspect is the fact that modern computing systems consist of a very large number of individual components. Even relatively closed systems such as software stacks for modern cars have about 100 million lines of codes and modern hardware also consists of large number of different components. As a consequence, many modules are not directly built by the manufacturers, but are bought from external sources. For systems handling sensitive data, this opens up a new attack vector in the form of *supply chain attacks*, where some of the imported modules are compromised:

How to construct a secure system if some of its parts are compromised?

A famous example of this is the DUAL_EC PRG [22, 21] that was constructed in such a way that it contained a backdoor, i.e., knowledge about the relation between two elliptic curve points, which allowed to recover the seeds from the outputs of the PRG. For cryptographic systems consisting of many modules M_i , the general setting is that some of these modules are replaced by compromised (or *subverted*) modules $\bar{M}_i$ in such a way that the use of $\bar{M}_i$ allows the subverter to break the cryptographic guarantees of the complete system. Furthermore, this should be done in such a manner that this subversion is not detected, i.e., M_i should be indistinguishable from $\bar{M}_i$. Such line of attacks were already presented by Young and Yung [40, 41], but were mostly treated as theoretical considerations. But the revelations of Snowden have shown that intelligence agencies were actively developing such attacks. Beginning with the work of Bellare et al. [5, 4], they have now become a well-studied line of attacks and the susceptibility of cryptographic systems to such attacks are now actively analyzed [24].

In general, there are two kinds of dangerous attacks that are unavoidable in a black-box manner. In so called *randomness subversion* attacks, the subverted module $\bar{M}_i$ samples randomness until the generated output contains some sensitive information accessible only to the subverter (e.g., the first 128 bit of output can be decrypted via AES and the first bit of the underlying message is the first bit of the sensitive information). The second kind of attack, called *trigger attacks*, only occurs if the subverted module $\bar{M}_i$ is given a maliciously chosen input on which it completely deviates from its usual behavior (e.g., it might send the sensitive information directly). Such attacks cannot be prevented in a purely black-box fashion [9]. And, while a trigger attack might be recognized a posteriori, randomness subversion attacks only generate valid outputs and can thus not even be detected after a possibly compromised run of the system.

As a protection, a number of different countermeasures were proposed. The two most common countermeasures are *watchdogs* [5, 37] and *cryptographic reverse firewalls* [36]. These watchdogs are small programs that compare the behavior of the possibly subverted module $\bar{M}_i$ to a reference implementation

(that is often much slower), either in an offline fashion before the deployment of the complete system or in an online fashion during the run of the system. By applying some kind of rerandomization to the inputs of the module (e.g., in the amalgamation model), watchdogs can be used to protect against trigger attacks.

The reverse firewalls are also small programs that are plugged into the communication of the possibly subverted system with another system. The original goal of the firewalls is to remove sensitive information from the output of the subverted system by rerandomizing these outputs and thus countering the randomness subversion attacks. To still allow correctness of the system, the incoming inputs (which might be answers to the rerandomized outputs) also need to be adapted similarly. In recent years, this adaption of the inputs has also been used to rerandomize the inputs to counter trigger attacks.

1.1 The Impersonation Problem

In the above discussion, we have so far only considered the scenario that one party, say Alice, is subversion corrupted. Now, Alice is suspicious and has thus installed a reverse firewall, protecting against the malicious effects of the subverted implementation that she is using. While this scenario is the common setting that reverse firewalls were designed for, using reverse firewalls also adds a new attack vector for the adversary. Reverse firewalls are untrusted by design and should never receive any secret information of Alice like the keys she is using. Therefore, one also has to consider the possibility that the reverse firewall itself is corrupted.

In case the reverse firewall is a separate component (e.g., a hardware device or a piece of software), Alice might be hesitant to use a reverse firewall that comes from the same provider (e.g., a hardware vendor or an app store) as the original system, as she would expect either both of them to be non-subverted or both of them to be subverted. Hence, she would most likely diversify by obtaining the original system and the reverse firewall from two different providers, say $Prov_S$ (for the original system) and $Prov_{RF}$ (for the reverse firewall). This gives us two further interesting situations: In the first one, $Prov_S$ provides a subverted implementation, but $Prov_{RF}$ is an honest implementation. This is exactly the situation for which reverse firewalls are designed and we thus obtain security. In the second situation, $Prov_S$ provides an honest implementation, but now the firewall provided by $Prov_{RF}$ is corrupted. Of course, as reverse firewalls do not have access to any secret information, even if the firewall is corrupted, no sensitive information can be leaked. However, a corrupted reverse firewall can still act maliciously. For example, as all incoming information to Alice go through the reverse firewall, it could simply cut off this information and thus isolate Alice. Similarly, all outgoing information from Alice also go through the reverse firewall, so the reverse firewall can also cut this type of communication.

In addition to this isolation, the reverse firewall can also send messages in the name of Alice to other parties, i.e., the reverse firewall can *impersonate* Alice. The reverse firewall can therefore act as a network adversary here. In the original game-based works on reverse firewalls, this has not led to issues, as these security games typically do not consider the problem of authentication. In contrast, models

closer to the real world such as Universal Composability need to address this issue of authentication. Usually, this is handled by working in a hybrid model assuming authenticated channels via the ideal functionality $\mathcal{F}_{\text{AUTH}}$ (Definition 3). However, as this functionality allows the communication between two explicitly stated parties, this would require either the communication channel to be directly between the other party and the reverse firewall (which is not desirable as whether or not a firewall is used should not matter to the other party), or the reverse firewall of Alice needs to communicate via this authenticated channel in the name of Alice. This ability is not covered by the ideal functionality $\mathcal{F}_{\text{AUTH}}$. This technical detail is often not mentioned in recent publications. Chakraborty et al. [20] address this issue of impersonating firewalls explicitly by allowing the firewall to send messages in the name of their main party via their newly introduced *sanitizable authentication* ideal functionality $\mathcal{F}_{\text{SAT}}$. However, this only makes the impersonate problem explicit, without solving it. Arnold et al. [1] already mention this impersonation problem and shortly discuss possible solution strategies. However, even the full version of their work [2] only presents solutions at a very high level lacking many details (which, as we discuss later on, have additional shortcomings). The issue of impersonation attacks is also mentioned by Liu et al. [32] as an important problem regarding the use of firewalls.

1.2 Technical Contribution

In this paper, we address a number of open questions regarding the design of UC-secure subversion-resilient protocols. Our two main achievements are:

- Generic compilers that transform *any* subversion-resilient protocol that realizes either the ideal commitment functionality $\mathcal{F}_{\text{COM}}$ (or the ideal coin toss functionality $\mathcal{F}_{\text{CT}}$, respectively) into a protocol that is both subversion and impersonation resilient and still realizes $\mathcal{F}_{\text{COM}}$ (or $\mathcal{F}_{\text{CT}}$, respectively).
- Identification of problems in existing UC subversion models that either are too strong ([20]) or too weak ([1]) and the development of a new, intermediate model avoiding these problems.

First, we show how the above mentioned impersonation problem can be handled, demonstrating that one can indeed securely deploy reverse firewalls without the danger of introducing a new attack surface. Previous works either were willing to accept this additional attack surface or ignored the issue altogether. We apply our techniques for commitments and coin toss to construct the first protocols that are subversion and impersonation resilient. The resulting protocols have a very high degree of *transparency*, a notion that guarantees that the parties do not need to be aware of the used firewalls and also allows easier, modular security proofs. While the protocols in [1] already support a form of transparency, we strengthen this significantly. But our new techniques are modular and not restricted to this: We show how to transform *any* subversion-resilient protocol that realizes $\mathcal{F}_{\text{COM}}$ (Definition 5) (or $\mathcal{F}_{\text{CT}}$ (Definition 6), respectively) into a protocol that is both subversion and impersonation resilient. This generic

transformation allows us to reuse existing protocols *and* existing security proofs. Our only required assumptions are a trusted key server and a public key that does not leak secrets. A practical method of generating the key pair (even after subversion) would be to generate it on a clean computer (that holds no secrets) and then send the public key to the key server and public/secret key pair to the main party.

In addition to being able to handle the impersonation problem present in nearly all previous work, we also closely analyzed the two main UC subversion frameworks [1, 20] that were already used in other works, e.g., [19, 33, 32, 34]: The model of Chakraborty et al. [20] aims to protect against both randomness subversion attacks and trigger subversions. To protect against the very strong trigger subversions, their firewalls need to also rerandomize incoming messages. We show that the model of Chakraborty et al. [20] does not allow the construction of certain secure functionalities, e.g., *there is no secure subversion-resilient string commitment* possible in their model⁵. The main issue is that firewalls can indeed defend against triggers from messages sent during the protocol, but not against protocol inputs which are directly given to the protocol by the environment $\mathcal{Z}$ ⁶. One can now use a secret high-entropy value a on which input the commitment behaves differently. In [20] a corruption with a subverted implementation is only allowed if this implementation is indistinguishable from the correct implementation for any efficient black-box test. However, if a is only known to $\mathcal{Z}$ and the subverted implementation, the subverted implementation is indistinguishable from an honest implementation for any efficient test. We elaborate on this problem of [20] in Section 4.

In contrast, the model of Arnold et al. [1] only protects against randomness subversion, i.e., attacks that replace the random tape in a careful manner. To model this, the authors introduce a `getRandomness` function, from which a protocol gets their randomness and the subversion is only allowed to manipulate this function. We show that this modeling allows for *trivially secure* protocols⁷: Suppose that a party draws a random value r_1 from the possibly subverted `getRandomness` function and is then given the uniformly distributed randomness r_2 , either by another party or by the firewall. The protocol is trivially secure if for the rest of the run, it uses $r_1 \oplus r_2$ as randomness and does not access `getRandomness` anymore. This weakness of [1] is also elaborated in Section 4. We remark here that such a trick was not used in the protocols of Arnold et al. [1] or Liu et al. [34].

As we discussed, existing UC subversion models are either too weak and allow for trivially secure protocols [1], or too strong and do not allow for secure protocols [20] for a wide range of functionalities. We remedy this situation by

⁵ We reached out to the authors of Chakraborty et al. [20] who acknowledged that there appears to be a problem and are currently looking deeper into this.

⁶ In the UC framework the environment $\mathcal{Z}$ is the entity that hands the protocol inputs to the parties, receives their outputs and gives instructions to the adversary.

⁷ We reached out to the authors of Arnold et al. [1] and they agreed with our observation.

presenting a new UC subversion modeling that does not suffer from the problems of [20] and strengthens [1] significantly, preventing “trivially secure” protocols.

1.3 Related Work

Reverse Firewalls The most common types of countermeasures to subversion attacks are reverse firewalls and watchdogs, where reverse firewalls usually aim to correct subverted implementations, whereas watchdogs mostly aim to detect them. Reverse firewalls were first introduced by Mironov and Stephens-Davidowitz [36]. Next to considering reverse firewalls for oblivious transfer (OT) and garbled circuits, they also showed how to construct them for two-party computation in the semi-honest setting against strong subversion adversaries that could even change the functionality of the underlying protocol. In [26], this was extended to the malicious setting but, as a consequence of strong impossibility results, only against functionality-maintaining adversaries. Follow-up work secured a lot of different primitives from subversion attacks using reverse firewalls, from signatures [3], over agreement protocols [26] and interactive proof systems [27] to active MPC [16, 17] and OT extensions [18].

Watchdogs When it comes to watchdogs, the line of work started with the work of Bellare et al. [5], and the notion was formalized by Russell et al. [37]. Watchdogs are given a reference implementation (which is typically too resource-intensive for real use). Offline watchdogs are used in an offline fashion before deployment of the system to perform a one-time check. They thus have a similar approach as our test used in the definition of subversion corruptions and are used to rule out “obviously” subverted algorithms that do not follow their specification at all. In contrast to this, online watchdogs have real-time access to the transcript of the communication. The watchdog model is typically combined with a non-black-box assumption (such as the split-program model or the trusted amalgamation model) that splits the algorithm into modules. Each such module can be subverted, but the amalgamation of those modules cannot. By leveraging such a model, various subversion-resilient primitives have been proposed, including one-way-permutations [37], pseudorandom generators [37], randomness generators [8, 37], public-key encryption schemes [8, 39], authenticated encryption [7], random oracles [38], and signature schemes [23, 37, 6].

Universal Composability To achieve security in the setting of multiple composed sessions, research is beginning to examine subversion attacks in the context of Universal Composability. As the latter does not directly allow this type of corruption, these works have to provide their own model that combines the two. Camenisch, Drijvers, and Lehmann [12] analyzed how to achieve anonymous attestation with subverted TPMs in UC. This work, however, does not use reverse firewalls to achieve security. The first ones that did consider reverse firewalls in the UC context were Li et al. [30], though they diverge from the original definition of reverse firewalls and re-define them including that such a firewall can also detect a subversion, in which case the reverse firewall will *alarm* and halt the whole protocol. One could thus describe it as a combination of the original reverse firewall definition and watchdogs. Chakraborty et al. [20] proposed a

UC framework that is both secure against *randomness subversion* as well as *input subversion* attacks. To handle the input subversion attacks, they need to drop the transparency notion of Mironov and Stephens-Davidowitz [36] and directly incorporate the reverse firewalls into their functionalities, requiring them to provide a new UC composition theorem adapted to subversion resilience. In contrast, Arnold et al. [1] chose to only consider *randomness subversion* which allowed them to use transparency and to stay in the plain UC framework. Note that the aforementioned models use authenticated channels to communicate, where these channels allow the reverse firewalls sitting between two parties to modify all incoming and outgoing messages.

Impersonation Usually the problem of impersonation attacks is only mentioned superficially, if at all. Chakraborty et al. [20] assume a sanitizable (i.e., malleable) authenticated transfer functionality throughout their work. In order to implement such a functionality given unauthenticated channels, they have to move to a “three-tier model” where each party also has an *operative component* coordinating communication between party, reverse firewall and network, which can only be honest or malicious. In this model, a reverse firewall cannot directly talk to another party, and only the reverse firewall sending messages to the operative component but not the party itself would be noticeable. Impersonation attacks would thus not be possible as long as this operative component is not corrupted. This model also provides a trivial solution to subversion attacks since the operative component could simply run the code of the party on behalf of it. While Arnold et al. tease solutions to impersonation attacks for commitment protocols in their full version [2], those are not really fleshed out. They missed the necessity of signing the session identifier and their idea can lead to compatibility issues with the message space of the commitment. These shortcomings are discussed in more detail in Section 3.1.

(P)AKE Subversion-resilient reverse firewall variants of authenticated key-exchange (AKE) and password-AKE (PAKE) have been also analyzed in the UC context [19, 33, 32]. While relying on the framework of [20] or [1], these primitives explicitly require unauthenticated communication channels. Hence, they have to modify the underlying framework. The case that the reverse firewall is maliciously corrupted is treated as a man-in-the-middle adversary and impersonation attacks are not considered explicitly and thus not specifically protected against. Note that in [19] the parties start the protocol with exchanging signature verification keys over the unauthenticated channel and then sign their subsequent messages. However, this still allows a malicious firewall to replace the signature verification key of its core with its own key and then execute the entire protocol, thereby impersonating the core.

Trojans Recently, the simulation-based work of Kumar et al. [29] introduced Trojan-resilient reverse firewalls, where the firewall itself contains a Trojan that can be either triggered by an incoming message or activate after a certain amount of time. Their solution is to divide the circuit of which the reverse firewall consists into multiple smaller sub-circuits that are connected by a so-called *master circuit*. The latter is a small, simple construction that is assumed to be honest and

	RF	UC	unauth. channels	Protection against impersonations
[20]	✓	✓	–	– three-tier model
[1]	✓	✓	–	– sketched-out approaches
[19, 33, 32]	✓	✓	✓	✗
[29]	✓	✗	N/A (no UC)	(✓) assum. continuously trusted hardware
Our work	✓	✓	✓	✓ assum. honest key pair

Table 1: Comparison of our work with SotA. The symbols are the following: ✓ denotes that this is supported by the model, ✗ denotes it is not, and finally – denotes that it was only superficially addressed.

incorruptible, ensuring that even if the sub-circuits are corrupted, the overall composition of these circuits still guarantees functional correctness. Comparing their assumption to ours (a trusted key server and a public key that does not leak secrets), one can see that the distribution of the used keys could be moved to a separate preprocessing phase; such a split is, however, not possible in the scenario of Kumar et al. [29].

We achieve the combination of subversion and impersonation resilience with the help of unique signatures. Ateniese, Magri, and Venturi [3] showed that unique signatures are secure against functionality-maintaining subversion attacks, as they can not be targeted by randomness subversion attacks. While the uniqueness property in the context of subversion resilience has mostly been used in regard to unique signatures (e.g., [25, 31]), it was also used for example for authenticated encryption with associated data [25].

2 Preliminaries

In this section, we give an overview of the concept of the (subversion-resilient) Universal Composability framework and relevant cryptographic primitives. Our used notations are introduced in a glossary in Appendix A.1.

2.1 Universal Composability

We give a short recap of the parts of the UC framework that are most important in this paper. In the UC framework [13], one analyzes the security of protocols, which usually consists of several parties that exchange messages. Each party gets its inputs from the so called *environment* machine $\mathcal{Z}$. The machine $\mathcal{Z}$ represents everything that is external to the protocol session under analysis. The environment also gives instructions to the adversary $\mathcal{A}$, such as the instruction to corrupt a party. The standard consequence of a corruption is that the adversary gains full control of the corrupted party (malicious corruption). However, in this paper, we additionally analyze security against so called *subversion corruptions*, where $\mathcal{A}$ exchanges the code of the party, but has no further control. The adversary also gets some information about the messages that all parties exchange (depending

on the communication channel that is used) and forwards this information to $\mathcal{Z}$. The execution of a protocol Π with $\mathcal{A}$ is called the *real world* execution.

Usually the security of a protocol Π is analyzed by comparing it to an ideal functionality $\mathcal{F}$. The ideal functionality acts like a trusted third party that gets the parties' inputs and gives them their outputs. The execution with the ideal functionality is called the *ideal world* execution. In the ideal execution, the adversary is replaced with a simulator Sim that interacts with $\mathcal{F}$. The task of Sim is to simulate the execution of the real protocol Π so that $\mathcal{Z}$ cannot distinguish between the real and ideal world. We then say that Π UC-realizes $\mathcal{F}$ if there is a simulator such that for all PPT environments, the real world execution of Π is indistinguishable from the ideal world execution of $\mathcal{F}$.

Note that parties in the real world can also communicate with ideal functionalities, which are then called *hybrid functionalities*. One example of this is the use of $\mathcal{F}_{\text{AUTH}}$, which parties use to exchange messages. The functionality $\mathcal{F}_{\text{AUTH}}$ models the properties of authenticated communication channels. The definitions of $\mathcal{F}_{\text{AUTH}}$ and other relevant ideal UC functionalities are provided in Appendix A.2.

One technical detail of the UC framework is that it splits each party into a *body* and a *shell*. The body contains the actual code of the protocol, whereas the shell contains the code that is responsible for, e.g., handling corruption instructions from $\mathcal{A}$. So whenever we describe a protocol, we describe the code for the body. However, to deal with the special corruption type of subversions, we will also make use of the shell formalism.

2.2 Subversion Resilience and Reverse Firewalls in UC

In the following, we give a brief recap of the two subversion UC frameworks of [20] and [1] in chronological order and note some of their differences.

Subversion UC of [20] (CMNV) The CMNV subversion model introduces *subversion* corruptions as a new corruption type. To perform a subversion corruption of a party $\mathcal{P}$, $\mathcal{A}$ sends a special message to the (shell of) $\mathcal{P}$ in which it specifies a new protocol code. From then on, the (body of) $\mathcal{P}$ executes this new code. The goal of this code is to leak information to $\mathcal{Z}$, which would allow $\mathcal{Z}$ to distinguish the real and ideal world. However, the CMNV model allows subversion corruption only with *specious* codes. The definition of speciousness in the model of CMNV [20, Def. 1, 3, 4] states that a subversion corruption is specious if it cannot be distinguished from an honest party by any efficient test. The definition explicitly allows so called triggers, i.e., once a specific message (the trigger) is received, the specious code is allowed to arbitrarily deviate from the protocol. This is made possible by the definition allowing the specious code and $\mathcal{Z}$ to share a high entropy value a , which is unknown to the test. For example, when the specious code receives a protocol message that is equal to a , the code could output a commitment to 0 (or secret data) as opposed to its actual value.

As a defense against the effects of subversion corruptions, each party is allowed to use firewalls. The firewall can modify all outgoing messages of a party in order

to prevent any information leakage. Also, in order to defend against triggers, firewalls are also supposed to modify the incoming messages to remove any potential trigger. The firewalls can also be corrupted by $\mathcal{A}$. The combination of corruptions of a party and their firewall is translated to one overall corruption type, which is treated as either honest or malicious.

The definition of *specious* subversion corruptions entails that one no longer formally prove that *there is a simulator such that for all PPT environments, the real and ideal world are indistinguishable*. Instead, one has to restrict the statement to only quantify over *specious* environments (i.e., environments that only use specious codes when doing subversion corruptions).

Subversion UC of [1] (ABMO) The ABMO subversion model also allows subversion corruptions that replace a party’s code with code specified by $\mathcal{A}$. However, unlike in the CMNV model, $\mathcal{A}$ is only allowed to modify the randomness generation. With this definition, one does not have to restrict $\mathcal{Z}$ to be specious, because the shell of the corrupted party can efficiently verify if a given code is specious and reject the corruption message if the code is not specious. This restriction of the subverted code still allows leaking secret information by embedding the secrets into the randomness. However, there are no triggers, as the code might run with manipulated randomness but cannot be changed itself.

To defend against the consequences of subversion corruptions, the parties are also allowed to have firewalls. Similarly to CMNV, the firewalls can also be corrupted and the combined corruption of a party and its firewall is translated to a classical corruption behavior of honest, semi-honest or malicious (though the exact details differ). The ABMO model relies on the firewalls to have two properties to simplify their security proofs: a firewall should be *transparent* and *anti-signaling*. Informally speaking, transparency means that the firewall is not noticeable to another party of the protocol (*outer transparency*) and does not require its party to change its code (*inner transparency*). If a firewall provides both transparency notions, it is called *strong transparency*. The other property, i.e., that the firewall is anti-signaling, describes that the firewall indeed counteracts the leakage of a subverted party. With the small change of restricting the environment to specious ones, we will also rely on these properties throughout this paper. For more information on the definitions of these properties, see Appendix A.3.

While in the CMNV model, subversion attacks are modeled in the ideal world, in the ABMO model, the subversion resilience of protocols is already proven between two real worlds: In one world, firewalls protect against subversion corruptions and in the other world, subversion corruptions are ignored and no firewall exists. Another difference to the CMNV model is that the firewalls in ABMO are not main parties of the protocol, i.e., they do not receive input from $\mathcal{Z}$. Additionally, only in the ABMO model does each party take a secret as additional input, which the subverted code can then try to leak.

2.3 Cryptographic Primitives

The most relevant cryptographic primitives of this work are unique signatures, non-interactive UC-secure commitments, and coin toss. In short, a signature scheme allows verifying the authenticity of a message. A unique signature scheme is a signature scheme where for each message, there is only *one* valid signature. A commitment scheme is a protocol between a committer and a verifier allowing the committer to first commit to a message in a hidden way, such that the message is fixed but the verifier does not learn it, and then opening (i.e., revealing) the message later on. Non-interactive UC-secure commitment schemes are commitment schemes where the only communication happening is from the committer to the verifier when first committing to a message and then sending the opening information. Finally, coin toss is an interactive protocol between two parties with the goal of generating a shared random coin. For more information on these three primitives, see Appendix A.3.

3 Compiler for Commitment and Coin Toss

In the ABMO model, the firewall was taken as an endpoint of the authenticated channel, because a firewall was modeled as a sub-party of a related main party. This modeling means that a firewall not only sanitizes an outgoing message from the party but also forwards the sanitized message in the name of the party. Thus, the firewall has the power to impersonate the party. More formally, in ABMO, the UC model assumes a direct channel between the main party and its sub-party (i.e., the firewall). The authenticated channel to another party is then modeled via the hybrid functionality of $\mathcal{F}_{\text{AUTH}}$ called by the firewall in the name of the main party. In contrast, in the CMNV model, the authors modeled the firewall as an own main party of the protocol, but had to introduce a new ideal functionality $\mathcal{F}_{\text{SAT}}$ representing the sanitizable version of an authenticated channel. While every party as well as its firewall communicates via $\mathcal{F}_{\text{SAT}}$, a message from the main party still has to pass through the firewall before being sent to another party. This again gives the firewall the explicit possibility to isolate and impersonate its related main party. To achieve subversion *and* impersonation resilience, we want to construct protocols without being in a (sanitizable) $\mathcal{F}_{\text{AUTH}}$ -hybrid UC world and thus rely on unauthenticated channels instead.

As discussed by Canetti in the ongoing description of the UC framework [14], $\mathcal{F}_{\text{AUTH}}$ can for example be realized by a digital signature combined with a public-key infrastructure. We make use of this and realize the authentication on compiler-protocol level: In our approach, the functional part of the protocol (e.g., the commitment) is directly combined with an authentication part that signs some meta data (e.g., a session identifier) to confirm both the content of the protocol (e.g., the commitment message) and that a party started the related communication. Thus, we make sure that there is no network adversary or firewall playing in between.

Regarding the signatures, similar to [32] we rely on a faithful shared key setting where no subversions took place. More formally, we will rely on the ideal

functionality for key registration with knowledge $\mathcal{F}_{\text{KRK}}$ (Definition 4) which we use to capture trusted key distribution. We show that the compiler protocol in the $\mathcal{F}_{\text{KRK}}$ -hybrid model UC-realizes the ideal functionality of the original protocol. For example, $\Pi_{\text{COM-comp}} \geq_{\text{UC}} \mathcal{F}_{\text{COM}}$ in $\mathcal{F}_{\text{KRK}}$ -hybrid model, when it is given that $\Pi_{\text{COM}} \geq_{\text{UC}} \mathcal{F}_{\text{COM}}$ in $\mathcal{F}_{\text{AUTH}}$ -hybrid model. One might think that this does not bring any advantages as we simply have two commitment protocols realizing the same ideal commitment functionality. However, by inserting the main authentication aspects already into the compiler protocol, we reduced the necessary hybrid from $\mathcal{F}_{\text{AUTH}}$ to $\mathcal{F}_{\text{KRK}}$.

3.1 Commitments without Authenticated Channel

In the following, we propose a compiler protocol for an authenticated commitment that internally applies unique signatures. The first signature, σ_1 , signs the session identifier sid known to both committer and verifier which is used to confirm that the verifier received the message from the committer. The second signature, σ_2 , signs the message m combined with the session identifier sid such that the verifier can later on verify that m is the message the committer put in the received opening information of the commitment c .

In Figure 1, we provide the simplified version of the carefully designed protocol. As designing a secure protocol in our setting is quite subtle, a discussion of the design choices and security aspects is given after the protocol description.

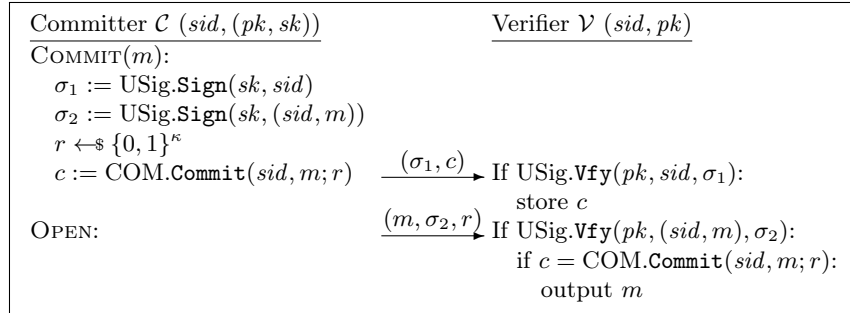


Fig. 1: A combined and simplified version of our protocol with a non-interactive commitment scheme.

Universal Composable Commitment Protocol without Authenticated Channel The following protocol $\Pi_{\text{COM-comp}}$ in Figure 2 realizes a two-party authenticated string commitment between a main-party committer $\mathcal{C}$ and a main-party verifier $\mathcal{V}$. Both the committer $\mathcal{C}$ and the verifier $\mathcal{V}$ have a protocol part for the commitment and one for the authentication (highlighted in gray boxes).

In the following, we describe the generalized protocol with black-box access to a non-interactive UC-secure commitment protocol COM. See Appendix A.4 for the interactions of $\mathcal{C}$ and $\mathcal{V}$ with COM. We assume that COM solely uses $\mathcal{F}_{\text{AUTH}}$ as communication channel between $\mathcal{C}$ and $\mathcal{V}$ (though other setup functionalities such as $\mathcal{F}_{\text{CRS}}$ are allowed).

$\Pi := \Pi_{\text{COM-comp}}$

Parameters:

- Functionality $\mathcal{F}_{\text{KRK}}$, see Definition 4.
- Protocol $\text{COM} := \Pi_{\text{COM}}^{\mathcal{F}_{\text{AUTH}}}$ in the $\{\mathcal{F}_{\text{AUTH}}, \mathcal{G}\}$ -hybrid model with committer $\mathcal{C}$ instance and verifier $\mathcal{V}$ instance as described in Appendix A.4,
- A unique signature scheme USig, see Definition 9.

Behavior of Party $\mathcal{C}$:

- Upon receiving $(\text{commit}, \text{sid}, m)$ from $\mathcal{Z}$, where $m \in \{0, 1\}^\kappa$,
 send **(register, sid)** to $\mathcal{F}_{\text{KRK}}$.
 Upon receiving **(registered, sid, $sk_{\mathcal{C}}$, $pk_{\mathcal{C}}$)**, get two unique signatures
 $\sigma_1 \leftarrow \text{USig.Sign}(sk_{\mathcal{C}}, \text{sid})$ and $\sigma_2 \leftarrow \text{USig.Sign}(sk_{\mathcal{C}}, (\text{sid}, m))$.
 Draw r randomly, get $c \leftarrow \text{COM}^{\mathcal{C}}.\text{Commit}(\text{sid}, m; r)$, store σ_2 ,
 and send **(send, sid, $\mathcal{C}$, (σ_1 , c))** to $\mathcal{V}$ via an unauthenticated channel
- Upon receiving **(open, sid)** from $\mathcal{Z}$, get $d \leftarrow \text{COM}^{\mathcal{C}}.\text{Open}(\text{sid})$, retrieve σ_2 ,
 and send **(send, sid, $\mathcal{C}$, (σ_2 , d))** to $\mathcal{V}$ via an unauthenticated channel.

Behavior of Party $\mathcal{V}$:

- Upon receiving **(sendOK, sid, $\mathcal{C}$, $\mathcal{V}$, (σ_1 , c))** from $\mathcal{C}$ via unauthenticated channel,
 send **(retrieve, sid, $\mathcal{C}$)** to $\mathcal{F}_{\text{KRK}}$.
 Upon receiving **(retrieved, sid, $\mathcal{C}$, $pk_{\mathcal{C}}$)**, store $(pk_{\mathcal{C}})$.
 If $\text{USig.Vfy}(pk_{\mathcal{C}}, \sigma_1, \text{sid}) = 1$:
 if $(\text{committed}, \text{sid}) \leftarrow \text{COM}^{\mathcal{V}}.\text{RcvCom}(\text{sid}, c)$:
 output **(committed, sid)** to $\mathcal{Z}$.
- Upon receiving **(sendOK, sid, $\mathcal{C}$, $\mathcal{V}$, (σ_2 , d))** from $\mathcal{C}$ via unauthenticated channel,
 retrieve $(pk_{\mathcal{C}})$.
 // if the commitment verification does not output a message, or if the signature
 does not verify, then the protocol outputs nothing.
 If $(\text{opened}, \text{sid}, m) \leftarrow \text{COM}^{\mathcal{V}}.\text{Vfy}(\text{sid}, d)$:
 if $\text{USig.Vfy}(pk_{\mathcal{C}}, \sigma_2, (\text{sid}, m)) = 1$: output **(opened, sid, m)** to $\mathcal{Z}$.

Fig. 2: Our commitment protocol in full detail. Gray boxes indicate code that is used for authenticating messages.

Design Choices and Discussion We briefly discuss and motivate the choices made during the design of the protocol. Further discussion of minor aspects is provided in Appendix B.1.

On signing the message outside of the commitment: An alternative to our protocol would have been to let the committer first sign the message and then commit to the signed message (this was suggested in [2]). We chose not to do this, because this would have required the message space of the commitment to match the one of the signed message, which, e.g., would have ruled out bit commitments. In contrast, our protocol works with commitments of arbitrary messages spaces.

On signing sid : The commitment functionality guarantees that the verifier outputs **committed** if and only if it received a commitment from the committer. Since in our setting, we do not have authenticated channels, a network attacker could trigger the **committed** output solely by sending a (fake) commitment to the verifier. In practice, this would be detrimental if the verifier takes a certain action only based on the event of receiving a commitment from the committer. In order to prevent this from happening, the committer sends a signature of sid which signals to the verifier that the committer actually did send a commitment. This was missed by [2] in the description of their idea for impersonation resilience.

Additionally, it is important that σ_2 is a signature on (sid, m) and not just m . Otherwise, an attacker can replay σ_2 from a previous session to perform the following attack: The attacker $\mathcal{A}$ passively observes one commitment session on message m . In the next session, $\mathcal{A}$ faithfully forwards the signature σ_1^{new} , but replaces the commitment with a (self-computed) commitment on m (for the current sid). For opening the commitment, $\mathcal{A}$ computes the opening information as usual and sends it together with σ_2^{old} from the old session. The verifier will then conclude that the committer has opened a commitment on message m , regardless of what the committer actually committed to.

On using $\mathcal{F}_{\text{KRK}}$: The use of $\mathcal{F}_{\text{KRK}}$ is sometimes considered to be a strong assumption, because $\mathcal{F}_{\text{KRK}}$ generates each party's key pair, which gives the UC simulator more power in the formal analysis. However, we want to stress that we do *not* use $\mathcal{F}_{\text{KRK}}$ to give the simulator more power. In fact, our simulator faithfully runs the code of $\mathcal{F}_{\text{KRK}}$ and does not use its knowledge of the committer's secret key. The reason we use $\mathcal{F}_{\text{KRK}}$ is that it let us avoid the case where a subverted committer tries to leak secret information via the signature public key. In practice, we expect the signature key to be a long-term key that has been generated before the committer was subverted. Nevertheless, in Appendix B.1 we discuss how to protect the committer even if it was subverted *before* generating the signature key pair, by rerandomizing the public key and signatures.

Security Analysis of Universal Composable Authenticated Commitment Protocol Next, we show that our protocol from Figure 2 UC-realizes the ideal commitment functionality $\mathcal{F}_{\text{COM}}$.

Theorem 1. *Let $\Pi_{COM-comp}$ be the authenticated commitment protocol defined in Fig. 2, which internally uses a non-interactive commitment protocol Π_{COM} . If $\Pi_{COM} \geq_{UC} \mathcal{F}_{COM}$ in the $\{\mathcal{F}_{AUTH}, \mathcal{G}\}$ -hybrid model and $USig$ is EUF-CMA-secure unique-signature, then we have that $\Pi_{COM-comp} \geq_{UC} \mathcal{F}_{COM}$ in the $\{\mathcal{F}_{KRK}, \mathcal{G}\}$ -hybrid model in the presence of malicious corruptions. That is, there exists a simulator $Sim_{\Pi_{COM-comp}}$ such that for all PPT environments $\mathcal{Z}$, $EXEC_{\Pi_{COM-comp}, \mathcal{A}, \mathcal{Z}}(\kappa, r) \approx EXEC_{\mathcal{F}_{COM}, Sim_{\Pi_{COM-comp}}, \mathcal{Z}}(\kappa, r)$, where $\mathcal{A}$ is the dummy adversary.*

In the following, we give the main idea of the proof for Theorem 1. The full proof is given in Appendix B.2.

Proof (sketch). As the underlying commitment Π_{COM} has to be UC-secure, the simulator Sim (given in Figure 3) uses the simulator of this protocol Sim_{COM} to simulate everything relevant as extractability and equivocalty for the commitment. The signature part is run by Sim itself and these are indistinguishable in both worlds, because the keys of the signature and messages are indistinguishable. In cases where $\mathcal{Z}$ would generate a signature forgery, we can rely on the EUF-CMA security of $USig$. The most interesting part is, that Sim has to handle and transfer all messages correctly between the internal Sim_{COM} , internal $\Pi_{COM-comp}$, $\mathcal{F}_{COM}$, and $\mathcal{Z}$, such that the unauthenticated direct communication is properly simulated.

3.2 Coin Toss without Authenticated Channels

In the following, we propose a compiler protocol for an authenticated coin-tossing protocol that internally signs and verifies the coin the parties agreed on. Therefore, both parties, $\mathcal{P}_0$ and $\mathcal{P}_1$, can be ensured they have the same coin in the end. In Figure 4, we provide the simplified version of the designed protocol.

Universal Composable Coin-Tossing Protocol without Authenticated Channel The following protocol $\Pi_{CT-comp}$ in Figure 5 realizes a two-party authenticated coin-tossing protocol between two main parties $\mathcal{P}_0$ and $\mathcal{P}_1$. Both parties have an internal protocol code for coin tossing and one for authentication (highlighted in gray boxes). For the signing part, we use unique signatures as defined in Definition 9. For coin tossing, both parties have black-box access to a UC-secure Π_{CT} , which is normally given in a setting of authenticated channels, i.e., in a $\mathcal{F}_{AUTH}$ -hybrid model. In our unauthenticated setting, we let the party, e.g., $\mathcal{P}_0$, run internally in a black-box way the respective code of Π_{CT} and every message that the internal Π_{CT} sends to $\mathcal{F}_{AUTH}$ is taken by $\mathcal{P}_0$ and sent via an unauthenticated channel. Vice versa, if $\mathcal{P}_0$ receives a message via the unauthenticated channel, it forwards the message to its internal instance of Π_{CT} as coming from $\mathcal{F}_{AUTH}$.⁸ We assume that Π_{CT} solely uses $\mathcal{F}_{AUTH}$ as

⁸ Note that this is also the reason why we cannot use $\mathcal{F}_{CT}$ instead of Π_{CT} as we have to carefully handle the incoming and outgoing messages via the unauthenticated channel instead of $\mathcal{F}_{AUTH}$.

```

Sim := SimΠCOM-comp

//  $\mathcal{F}_{\text{KRK}}$  handling:
- On (register, sid) from  $\mathcal{Z}$  in the name of corrupted party  $\mathcal{P}_i$ : imitate  $\mathcal{F}_{\text{KRK}}$ .
- On (retrieve, sid) from  $\mathcal{Z}$  in the name of party  $\mathcal{P}_i$ : imitate  $\mathcal{F}_{\text{KRK}}$ .

// Hybrid functionality  $\mathcal{G}$  handling:
- On call from  $\mathcal{Z}$  for  $\mathcal{G}$ : forward it to  $\text{Sim}_{\text{COM}}$  and give response back to  $\mathcal{Z}$ .

Case: Both parties honest
- On (deliver committed, sid) from  $\mathcal{F}_{\text{COM}}$ :
  • Generate  $(sk_{\mathcal{C}}, pk_{\mathcal{C}})$  and get  $\sigma_1 \leftarrow \text{USig.Sign}(sk_{\mathcal{C}}, sid)$ .
  • Instruct internal  $\text{Sim}_{\text{COM}}$  with (deliver committed, sid) in the name of  $\mathcal{F}_{\text{COM}}$ .
  • Get (send, sid,  $\mathcal{C}$ ,  $\mathcal{V}$ , c) from  $\text{Sim}_{\text{COM}}$  as its output to  $\mathcal{Z}$ .
  • Store  $sk_{\mathcal{C}}$  and output (send, sid,  $\mathcal{C}$ ,  $\mathcal{V}$ ,  $(\sigma_1, c)$ ) to  $\mathcal{Z}$ .

- On (deliver opened, sid, m) from  $\mathcal{F}_{\text{COM}}$ :
  • Retrieve  $(sk_{\mathcal{C}})$  and get  $\sigma_2 \leftarrow \text{USig.Sign}(sk_{\mathcal{C}}, (sid, m))$ .
  • Instruct  $\text{Sim}_{\text{COM}}$  with (deliver opened, sid, m) in the name of  $\mathcal{F}_{\text{COM}}$ .
  • Get (send, sid,  $\mathcal{C}$ ,  $\mathcal{V}$ , d) from  $\text{Sim}_{\text{COM}}$  as its output to  $\mathcal{Z}$ .
  • Store m and
  • Output (send, sid,  $\mathcal{C}$ ,  $\mathcal{V}$ ,  $(\sigma_2, d)$ ) to  $\mathcal{Z}$ .

- On (sendOK, sid,  $\mathcal{C}$ ,  $\mathcal{V}$ , msg) from  $\mathcal{Z}$ :
  • Instruct  $\mathcal{V}$  of  $\Pi_{\text{COM-comp}}$  with (sendOK, sid,  $\mathcal{C}$ , msg).
  • Get output from  $\mathcal{V}$  of  $\Pi_{\text{COM-comp}}$  as its output to  $\mathcal{Z}$ .
  • If output = (committed, sid):
    * send (deliver ok committed, sid) to  $\mathcal{F}_{\text{COM}}$ .
  • Else-if output = (opened, sid,  $\bar{m}$ ):
    * If  $\bar{m} = m$ : // if m has not been stored before, do nothing.
    * send (deliver ok opened, sid) to  $\mathcal{F}_{\text{COM}}$ .

Case: Committer malicious, Verifier whatever
- On (corrupt, sid) from  $\mathcal{Z}$  to  $\mathcal{P}_i$ :
  forward (corrupt, sid,  $\mathcal{P}_i$ ) to  $\text{Sim}_{\text{COM}}$  and  $\mathcal{F}_{\text{COM}}$ .

//  $\mathcal{Z}$  has to confirm the delivery of its own messages.
- On (send, sid,  $\mathcal{C}$ , msg) from  $\mathcal{Z}$  in the name of  $\mathcal{C}$ :
  send (send, sid,  $\mathcal{C}$ ,  $\mathcal{V}$ , msg) to  $\mathcal{Z}$  as backdoor message.

// Commit-Phase:
- On (sendOK, sid,  $\mathcal{C}$ ,  $\mathcal{V}$ , msg =  $(\sigma, c)$ ) from  $\mathcal{Z}$ :
  • Retrieve  $(pk_{\mathcal{C}})$ .
  • If  $\text{USig.Vfy}(pk_{\mathcal{C}}, \sigma, sid) = 1$ :
    * Instruct  $\text{Sim}_{\text{COM}}$  with (sendOK, sid, c) in the name of  $\mathcal{Z}$ .
    * Get (commit, sid, m) from  $\text{Sim}_{\text{COM}}$  as its output to  $\mathcal{F}_{\text{COM}}$  in the name of corrupted  $\mathcal{C}$ .
    * Store m.
    * Send (commit, sid, m) to  $\mathcal{F}_{\text{COM}}$  in the name of corrupted  $\mathcal{C}$ .

// Opening-Phase
- On (sendOK, sid,  $\mathcal{C}$ ,  $\mathcal{V}$ , msg =  $(\sigma, d)$ ) from  $\mathcal{Z}$ :
  • Retrieve m and  $pk_{\mathcal{C}}$ .
  • If  $\text{USig.Vfy}(pk_{\mathcal{C}}, \sigma, (sid, m)) = 1$ :
    * Instruct  $\text{Sim}_{\text{COM}}$  with (sendOK, sid, d) in the name of  $\mathcal{Z}$ .
    * If  $\text{Sim}_{\text{COM}}$  returns (open, sid) as message to  $\mathcal{F}_{\text{COM}}$  in the name of corrupted  $\mathcal{C}$ :
      * Send (open, sid) to  $\mathcal{F}_{\text{COM}}$  in the name of corrupted  $\mathcal{C}$ .

```

Fig. 3: Simulator for our commitment protocol compiler from Figure 2.

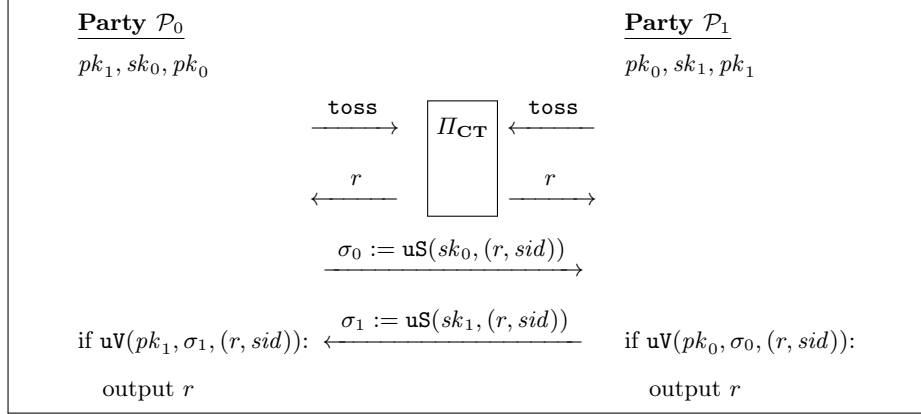


Fig. 4: A simplified version of our combined protocol of coin toss and unique signatures. To shorten notation, we write $\text{uS} := \text{USig.Sign}$ and $\text{uV} := \text{USig.Vfy}$.

communication channel (though other setup functionalities such as $\mathcal{F}_{\text{CRS}}$ are allowed). E.g., in the case where Π_{CT} is in the $\mathcal{F}_{\text{COM}}$ -hybrid model one has to insert the commitment protocol first so that the resulting protocol is only in the $\mathcal{F}_{\text{AUTH}}$ -hybrid model.

Security Analysis of Universal Composable Authenticated Coin-Tossing Protocol Next, we will show that our protocol from Figure 5 UC-realizes the ideal coin-tossing functionality $\mathcal{F}_{CT}$.

Theorem 2. *Let $\Pi_{CT\text{-comp}}$ be the authenticated coin-toss protocol as defined in Figure 5, which internally uses a coin-tossing protocol Π_{CT} . If $\Pi_{CT} \geq_{UC} \mathcal{F}_{CT}$ in the $\{\mathcal{F}_{\text{AUTH}}, \mathcal{G}\}$ -hybrid model and USig is EUF-CMA-secure unique-signature, then we have that $\Pi_{CT\text{-comp}} \geq_{UC} \mathcal{F}_{CT}$ in the $\{\mathcal{F}_{\text{KRK}}, \mathcal{G}\}$ -hybrid model in the presence of malicious corruptions. That is, there exists a simulator $\text{Sim}_{\Pi_{CT\text{-comp}}}$ such that for all PPT environments $\mathcal{Z}$, $\text{EXEC}_{\Pi_{CT\text{-comp}}, \mathcal{A}, \mathcal{Z}}(\kappa, r) \approx \text{EXEC}_{\mathcal{F}_{CT}, \text{Sim}_{\Pi_{CT\text{-comp}}}, \mathcal{Z}}(\kappa, r)$, where $\mathcal{A}$ is the dummy adversary.*

Proof Idea: The case where one party is honest and the other is malicious is rather straightforward. Because $\Pi_{CT} \geq_{UC} \mathcal{F}_{CT}$, there is a corresponding simulator $\text{Sim}_{\Pi_{CT}}$ that simulates Π_{CT} while talking to $\mathcal{F}_{CT}$. Our simulator Sim uses $\text{Sim}_{\Pi_{CT}}$ to simulate Π_{CT} . The fact that Π_{CT} has authenticated channels, while $\Pi_{CT\text{-comp}}$ does not, is not a problem, because the malicious party can send arbitrary messages anyways. The main thing that Sim needs to do is to feed the correct messages into $\text{Sim}_{\Pi_{CT}}$ and simulate the signature exchange (since that is not part of Π_{CT}).

One could assume that the case of two honest parties works analogously. Unfortunately, this is not the case, because, due to the lack of authenticated

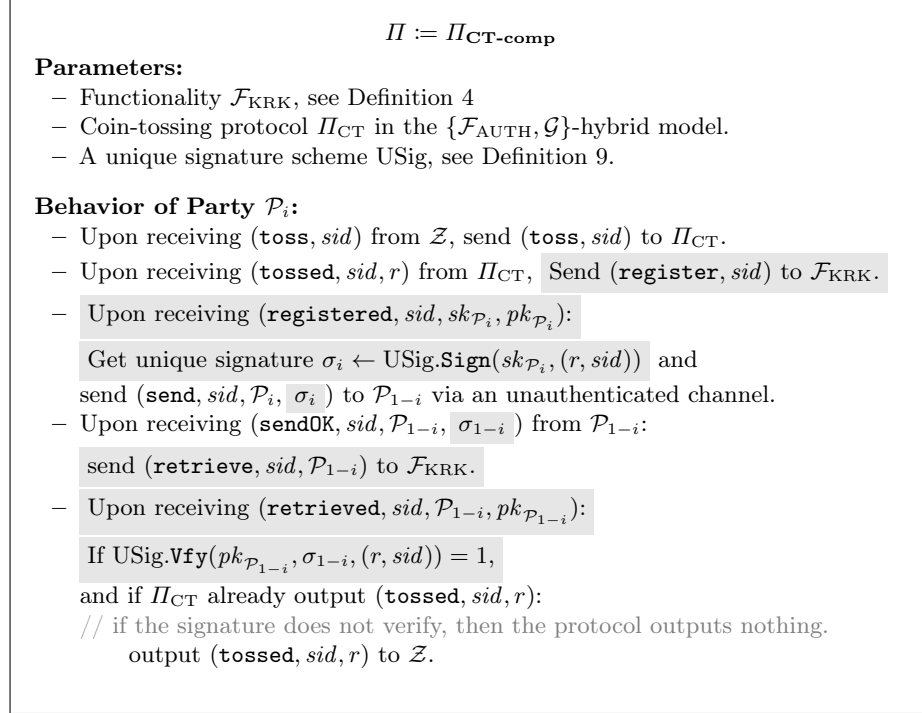


Fig. 5: Our coin-tossing protocol in full detail. Gray boxes indicate code that is used for authenticating messages.

channels, $\mathcal{Z}$ can modify messages between the honest parties. When $\mathcal{Z}$ modifies a message, Sim cannot map this to an instruction for $\text{Sim}_{\Pi_{\text{CT}}}$, because $\text{Sim}_{\Pi_{\text{CT}}}$ lives in a world with authenticated channels and only expects a yes/no instruction for whether to deliver a message. We solve this problem by letting Sim tell its internal $\text{Sim}_{\Pi_{\text{CT}}}$ that $\mathcal{P}_1$ is malicious. Hence, $\text{Sim}_{\Pi_{\text{CT}}}$ will simulate messages for the honest $\mathcal{P}_0$, but not for $\mathcal{P}_1$. Therefore, Sim simulates the honest $\mathcal{P}_1$, which is possible, because $\mathcal{P}_1$ has no secret inputs. The simulator Sim then routes the messages between $\text{Sim}_{\Pi_{\text{CT}}}$, the simulated $\mathcal{P}_1$, $\mathcal{Z}$ and $\mathcal{F}_{\text{CT}}$, so that everything works out.

In Appendix C.1 we provide more detail on the routing of the messages, in Figure 8 the full simulator, and in Appendix C.2 the proof of Theorem 2.

4 Revisiting Subversion UC

As it turns out, the two subversion UC frameworks of ABMO [1] and CMNV [20] both suffer from different problems. The ABMO framework is too weak for certain realistic threat models, whereas the CMNV model is too strong so that many

useful protocols such as string-commitments are impossible in the model. We start by explaining and discussing these problems. We then define a new notion of subversion resilience in UC and describe how to integrate this notion into the frameworks of ABMO and CMNV to solve their problems. Furthermore, in Appendix D.1 we state some lemmas that are useful for working with our new model.

4.1 Problems with the Definitions of CMNV [20] and ABMO [1]

Here we describe problems with the subversion frameworks of CMNV and ABMO. A short recap of both frameworks can be found in Section 2.2.

Problems with CMNV [20]: The speciousness definition of [20] has the following problems⁹: First of all, the definition allows triggers via the protocol input, against which a firewall cannot defend. Secondly, *leakage* can also happen via the output, as the value that a subverted party is supposed to output can be used as a trigger as well.

In more detail, in the UC framework, the protocol input comes directly from the environment $\mathcal{Z}$. As this type of communication is not passed through a firewall, there is no way for a firewall to remove the trigger. As a concrete example, consider the following code of a specious committer (with hardcoded high entropy a also known to $\mathcal{Z}$) in a commitment protocol:

- On input (`commit`, sid , m):
 - if $m = a$: commit to $0 \dots 0$
 - else: commit to m
- On (`open`, sid): open the commitment

This code is specious, because no test can guess a and, thus, the code will always behave honestly in the test setting. However, the environment can distinguish the real and ideal world with the following steps:

- Do a subversion corruption of the committer with the above code.
- Give (`commit`, sid , $m = a$) as input to the committer.
- Instruct the firewall to sanitize the commitment without affecting the committed value.
- Give (`open`, sid) as input to the committer.

In the real world, the verifier will output (`opened`, sid , $0 \dots 0$) as this is what the committer committed to. Now in the ideal world, the verifier will output (`opened`, sid , $m = a$), as there is no subversion in the ideal world and the simulator has no way of influencing the committed value.

The second problem is that a subverted party can take the value it is supposed to output and use it as a trigger. As a concrete example, consider a subverted verifier in a commitment protocol. The subverted verifier honestly executes the

⁹ We reached out to the authors of Chakraborty et al. [20] who acknowledged that there appears to be a problem and are currently looking deeper into this.

protocol, except that before outputting $(\text{opened}, \text{sid}, m)$ to $\mathcal{Z}$, it checks if $m = a$. If so, it outputs $0 \dots 0$. Otherwise, it outputs m honestly. Now, $\mathcal{Z}$ can call the committer with $(\text{commit}, \text{sid}, a)$ and then $(\text{open}, \text{sid})$. In the real world, the verifier will output $0 \dots 0$ and in the ideal world, the verifier will output $m = a$, as the simulator has no way of influencing the value of the commitment.

Together, these two problems rule out a large class of protocols.

Problems with ABMO [1]: We now turn towards a problem with the subversion notion of ABMO [1]¹⁰. In a nutshell, we believe that the notion is too weak and allows to prove protocols as subversion resilient which should not be considered subversion resilient in practice (though this depends on one’s threat model).

The ABMO framework allows subverted code to only modify the randomness of the implementation. While this captures many real world subversion attacks, the notion of “*only manipulate randomness*” can be abused by protocol designers to design protocols that are *trivially* subversion resilient. As an example, consider the following generic trick: Before the actual protocol starts, a party draws its own randomness r_1 (which is potentially manipulated) and then gets r_2 by receiving a random string from its counterparty (which is additionally rerandomized by the firewall). Then the party computes $r := r_1 \oplus r_2$ and uses r as randomness for the rest of the protocol. Now, the potentially manipulated r_1 has no effect, as r is uniformly random because r_2 is uniformly random¹¹. Also the protocol retains all its other security guarantees, because if the main party is honest, r is guaranteed to be uniformly random as well. Therefore, prepending the above behavior to a protocol would trivially make it subversion resilient in the ABMO framework. However, in practice a subverted code could simply manipulate r in order to leak information about the secret, without increasing its risk of getting caught.

We note two points. First, the authors of ABMO [1] do *not* use the above trick and, thus the protocols presented in [1] are presumably subversion resilient in more realistic settings. Second, there exists a (somewhat reasonable) threat model under which the notion of ABMO is *not* too weak. If the main concern is that the operating system’s randomness generation is manipulated so that all programs will use manipulated randomness, then this is exactly captured by the ABMO model. Furthermore, in that setting, the above *trick* that incorporates external randomness is actually an effective countermeasure against the subversion. If, however, one also worries that the implementation of, e.g., the commitment scheme itself is manipulated, then our criticism applies.

¹⁰ We reached out to the authors of Arnold et al. [1] and they agreed with our observation.

¹¹ Since $\mathcal{A}$ sees the counterparty’s random string, the entropy in r_2 comes from the firewall. Note that one does not consider the case where the main party is subverted and the firewall is *not* honest, because then leakage of the secret is unavoidable.

4.2 Our Definition of Subversion Resilience

In this section, we present our notion of subversion resilience and how to model it in the UC framework. Finally, we describe how to integrate our notion in both the CMNV and the ABMO model to obtain full-fledged subversion UC frameworks.

We focus on a subversion setting without triggers for two reasons. First, allowing triggers would bring back the problems of the CMNV model that we discussed in Section 4.1. Second, we believe that there is also practical justification for disallowing triggers. Assume a party receives a trigger and then leaks secret information by deviating from the protocol in a detectable manner. This may seem like a plausible attack vector. However, an adversary cannot use this trigger at scale while remaining undetected, because every use of the trigger has a certain risk of being caught, which would likely result in the corrupted implementation being fixed. Not covering triggers is also backed by the observation that all practical subversion attacks merely modify the protocol’s randomness ([4, 40, 5, 9, 3]).

As we have seen above, the restriction that a subverted code is only allowed to modify the protocol randomness can be abused to achieve trivial subversion resilience. To avoid this, it seems necessary to allow a subverted code to do anything as long as no efficient black-box test T can distinguish between running the subverted code and running an honest code C . This notion was proposed by the authors of the CMNV model [20] and referred to as *specious* corruptions.

A UC subversion notion that allows specious corruptions, but disallows trigger has to resolve the following tension. If the subverted code $\tilde{C}$ has a high entropy shared secret a with the environment $\mathcal{Z}$, then $\mathcal{Z}$ can use a as a trigger, negating our goal of disallowing trigger. If $\tilde{C}$ does not have a high entropy shared secret with $\mathcal{Z}$, then the test T is essentially as powerful as $\mathcal{Z}$. So when the $\tilde{C}$ tries to leak information, it is caught by T . We resolve this tension by giving $\tilde{C}$ access to a high entropy secret, but only handing that secret to $\mathcal{Z}$ when the protocol is over. Thereby, $\tilde{C}$ is able to leak information to $\mathcal{Z}$ (e.g., via using the secret as randomness in a commitment) while remaining undetected by T , giving us a meaningful security notion. At the same time $\mathcal{Z}$ cannot use the secret as a trigger, because it only learns the secret after $\tilde{C}$ has been deactivated.

Intuition of our Subversion Notion The core idea for our notion is the following: We let a distinguisher try to distinguish a subverted execution from an honest execution. To allow the subverted code to leak information while being specious, we must make the distinguisher more powerful than the test. Therefore, when the protocol ends, the protocol transcript *and* the secret are given to the distinguisher. Note that since during the protocol execution, nobody except for the subverted party knows the secret, it cannot be used as trigger.

This intuition can be expressed a bit more formally (though, not yet in the UC framework) as follows. We only consider subverted codes that are indistinguishable from honest ones for any efficient black-box test. A distinguisher has to distinguish between the two following cases:

- *Subverted execution*: At the beginning of the protocol, the adversary (which is a different party than the distinguisher) can decide which parties to subvert and which parties to corrupt maliciously. Each subverted party samples a uniformly random secret, to which the subverted code has read access. Then, the protocol is executed and once it is over, the distinguisher receives the protocol transcript and the secrets of the subverted parties.
- *Honest execution*: At the beginning of the protocol, the adversary can decide which parties to subvert and which parties to corrupt maliciously. If the adversary decides to subvert a party, this is recorded, but *the party remains honest*. Then, the protocol is executed and once it is over, the distinguisher receives the protocol transcript, and a random value for each of the parties that the adversary wanted to subvert.

We now describe how the above can formally be captured in the UC framework.

Formal Definition Our notion only allows subversion corruptions with a *specious* code. Let C be the code of a party of a UC protocol. Let $\tilde{C}$ be any code with the same input-output interfaces as C . We say that $\tilde{C}$ is *specious* (relative to C) if $\tilde{C}$ and C are indistinguishable for any efficient (non-uniform) test T . Following [20], we write $T^{\tilde{C}}$ to denote that T has non-rewinding black-box access to $\tilde{C}$, where $\tilde{C}$ has access to randomness about which T does not get any information. When $\tilde{C}$ makes a subroutine call to another protocol or ideal functionality, then T sees this call and is responsible for answering. More concretely, we define the following game along the lines of the speciousness definition of [20]:

- Sample a uniformly random secret sec , to which the code gets read access.
- Sample $b \leftarrow_{\$} \{0, 1\}$ and
 - if $b = 0$, then run $T^{\tilde{C}}$ and get a guess $b' \in \{0, 1\}$;
 - if $b = 1$, then run T^C and get a guess $b' \in \{0, 1\}$.
- Output $b \oplus b'$.

Let $\text{Test}_T^{C, \tilde{C}}$ denote the probability that the game outputs 0 (i.e., that T guessed correctly).

Definition 1 (Speciousness). *A code $\tilde{C}$ is specious (relative to a code C) if for all non-uniform PPT tests T , it holds that $\text{Test}_T^{C, \tilde{C}} - \frac{1}{2}$ is negligible.*

Note that the secret sec in this game does not have any function, but as the code $\tilde{C}$ in our UC setting will have read access to sec , for syntactical correctness, we also have it in this game. Also note that the above definition of speciousness is almost identical to CMNV [20, Def. 1]. The difference is that we define speciousness of codes $\tilde{C}$, whereas [20] defined speciousness of distributions, from which the code is then sampled. Their definition allowed the code to contain a hardcoded secret/trigger that the test does not know (but $\mathcal{Z}$ does). As we do *not* want to allow trigger attacks, we chose to define speciousness directly for codes.

Our subversion notion in the UC setting: Following ABMO and CMNV we only consider static corruptions. In the (UC) real world we allow subversion corruptions. In more detail, when the shell of a party receives the instruction $(\text{subversion-corruption}, \tilde{C})$, it chooses a uniformly random secret to which the body (i.e., $\tilde{C}$) has read access. Then, it replaces the honest code C with $\tilde{C}$. Furthermore, (the shell of) a party $\mathcal{P}$ responds to an input **reveal-secret** in the following way: First it sends $(\text{reveal-secret}, \mathcal{P})$ to $\mathcal{A}$. If the $\mathcal{P}$ is subverted, then the shell returns the secret to the environment. If the $\mathcal{P}$ is not subverted, the shell returns a freshly chosen random value of the same length as the secret. From then on, the shell will ignore all messages. In other words: After processing the **reveal-secret** instruction, the party is deactivated. Whenever a protocol allows subversion corruptions and implements the **reveal-secret** interface as described above, we write Π^s (instead of Π).

In the (UC) ideal world, subversion corruptions are ignored by the ideal functionality (unless otherwise specified). When the ideal functionality receives **reveal-secret** as input from a party, it notifies the simulator and then returns a random value of the same length as the secret to $\mathcal{Z}$. From then on, the ideal functionality will neither accept input nor produce output for that party. Whenever an ideal functionality ignores subversion corruptions and implements the dummy **reveal-secret** interface as described above, we write $\mathcal{F}^r$ (instead of $\mathcal{F}$). Because it will be useful later we also use the notation Π^r to refer to a (UC) real world protocol in which the shells of all parties ignore subversion corruptions and handle **reveal-secret** messages the same way as $\mathcal{F}^r$. Furthermore, the **reveal-secret** interface of the main party also returns the secrets of the sub-parties¹². Unfortunately, this gives information about the sub-party structure to $\mathcal{Z}$, which would allow $\mathcal{Z}$ to distinguish real and ideal world (as an ideal functionality does not have sub-parties). Therefore, we allow the simulator to specify a fake sub-party structure for party $\mathcal{P}$ when it is notified with $(\text{reveal-secret}, \mathcal{P})$. Second, when subverting a party that has sub-parties, one has to specify the code for all the sub-parties and all those codes need to be specious when considered in isolation. We will write *specious* environment (or simulator) to denote environments (or simulators) that only use specious codes when doing subversion corruptions.

We say that a protocol Π srUC-realizes an ideal functionality $\mathcal{F}$ (or protocol Φ) if Π UC-realizes $\mathcal{F}$ (or Φ) with the restriction that we only quantify over specious environments and simulators (written as $\Pi \geq_{\text{srUC}} \mathcal{F}$). Restricting $\mathcal{Z}$ to be specious is necessary, because otherwise it would be impossible to get any meaningful guarantees for useful protocols. Restricting the simulator to be specious is necessary, because otherwise transitivity of srUC-emulation would not hold.

¹² Sub-parties need to have secrets, otherwise the notion of subversion resilience could be fulfilled trivially by simply outsourcing the entire protocol to a sub-party. However, sub-parties also cannot have the same secret as the main-party, because then the main-party could use the secret as trigger for the sub-party.

Discussion of Our Subversion Notion

How does this notion guarantee subversion resilience? This notion allows two options to show the subversion resilience of a given protocol. The first is to show that it realizes an ideal functionality that ignores subversion corruption messages and returns freshly generated random secrets as a response to a **reveal-secret** input. The execution in the ideal world is independent of the secret. Thus, if the real protocol execution is indistinguishable from the ideal execution, this means that the subverted code in the real world was unable to leak useful information to $\mathcal{Z}$. Formally, to show subversion resilience, one has to prove that $\Pi^s \geq_{\text{srUC}} \mathcal{F}^r$.

The other option is to show that a protocol Π' that allows subversion corruptions is UC-emulating a protocol Π which does not allow them. This is particularly useful if Π' is the combination of Π and a firewall RF and it has already been shown that $\Pi \geq_{\text{UC}} \mathcal{F}$. Then we can simply show that $\Pi^s + RF \geq_{\text{srUC}} \Pi^r$ and then use $\Pi \geq_{\text{UC}} \mathcal{F}$ together with Lemma 3-6 to get $\Pi^s \geq_{\text{srUC}} \mathcal{F}^r$.

How does this notion disallow triggers? If a trigger would be hardcoded in $\tilde{\mathcal{C}}$, it could be found by an efficient (non-uniform) test, making $\tilde{\mathcal{C}}$ non-specious. So instead, a high-entropy shared value between $\tilde{\mathcal{C}}$ and $\mathcal{Z}$ could be used as a trigger, as was done in [20]. However, this is not possible in our notion, because during the protocol execution, the secret is only known to the subverted party and not to $\mathcal{Z}$. As soon as $\mathcal{Z}$ learns the secret, the subverted party is deactivated and, thus, cannot respond to any trigger anymore.

Integrating Our Notion with ABMO and CMNV We can use this subversion resilience notion to adapt the models of ABMO and CMNV.

Integration with ABMO [1]: To integrate our notion of UC subversion resilience into the framework of ABMO [1] (which we then call the *new ABMO model*), we need to do the following: Recall that the original ABMO model only allowed subverted codes to manipulate the randomness. In contrast, we now allow subversions with all codes that are specious with regard to Definition 1. In the original ABMO model, each party has an interface for $\mathcal{Z}$ to hand a secret to that party. In our new ABMO model, the secret of each party is generated randomly by its shell and given to $\mathcal{Z}$ as response to a call to the **reveal-secret** interface as described in Section 4.2. When realizing ordinary ideal functionalities, we thus also have to add the dummy **reveal-secret** interface to $\mathcal{F}$ (which we then call $\mathcal{F}^r$). Contrary to [1], we now have to restrict the notion of srUC-emulation/realization to only quantify over environments that adhere to the speciousness definition when doing subversion corruptions. However, we can still follow [1] and prove subversion resilience already in the (UC) real world. For this, we have to show that the protocol Π'^s that accepts subversion corruption and has firewalls srUC-emulates the protocol Π^r that does not accept subversion corruptions.

Integration with CMNV [20]: To integrate our notion of UC subversion resilience into the framework of CMNV [20] (which we then call the *new CMNV model*), we need to do the following: First, we replace their definition of speciousness with Definition 1. As in CMNV, the shells of the parties already accept subversion corruption, we then need to add to the shells the generation of the secret upon subversion corruption and the handling of the **reveal-secret** instruction as described in Section 4.2. Finally, the **reveal-secret** interface needs to be added to the ideal functionalities. This can be achieved by adopting our notion of $\mathcal{F}^r$. Everything else remains unaffected. In particular, in this new model, the firewalls can still be main parties of the protocol and receive input from $\mathcal{Z}$ to, e.g., sanitize the input of the protocol.

For the rest of the paper, we will use the new ABMO model as a basis, because ABMO [1] explicitly allow showing subversion resilience in the real world, without the need for any ideal functionality. This will turn out to be a very useful tool for our analysis in Section 5. We note, however, that the new ABMO model and the new CMNV model are (on a reasonable level of abstraction) almost identical. In Appendix D.1, we state a few lemmas which are useful when working with our model of subversion resilient UC (i.e., the integration of our subversion notion in ABMO). The lemmas will be applied in Section 5.

5 Subversion- and Impersonation-Resilient Protocols

Now that we have a coherent notion of subversion resilience in the UC framework, we can use the preparatory work of Section 3 to construct subversion- and impersonation-resilient protocols for commitments and coin toss. We first start with considering commitments (Section 5.1). The construction and analysis of the coin-tossing protocol (Section 5.2) is then in large parts analogous. We analyze our protocols using our new notion of subversion resilience integrated into the ABMO framework as described in Section 4.2. The results then carry over to our new notion in the CMNV framework, as well as to the *original* ABMO framework with minor modifications. The impersonation resilience of our protocols is reflected by the fact that an honest party with a malicious firewall is treated as an honest party (instead of a malicious party). This is achieved by our construction ensuring that a malicious firewall can do no more harm than the network adversary. The updated corruption transition table can be seen in Table 2.

5.1 Subversion- and Impersonation-Resilient Commitment

In Figure 6, we present our construction of a subversion- and impersonation-resilient commitment protocol. Recall the notion of Π^s and $\mathcal{F}^r$ that we introduced in Section 4.2. We start with a protocol Π_{COM} in the $\mathcal{F}_{\text{AUTH}}$ -hybrid model that UC-realizes $\mathcal{F}_{\text{COM}}$. Furthermore, we need a transparent and anti-signaling firewall RF for Π_{COM} (here we still allow RF to be the communication endpoint of $\mathcal{F}_{\text{AUTH}}$). This means that Π_{COM} together with RF srUC-realizes $\mathcal{F}_{\text{COM}}^r$, though

Table 2: Different corruption combinations and their composed classic corruption behavior. The lines in gray show the important results of this work, i.e., subversion and impersonation resilience.

Party	Firewall	Composed Classic Behavior
honest	honest/semi-honest	honest
honest	malicious	honest
malicious	honest/malicious	malicious
subversion	honest	honest

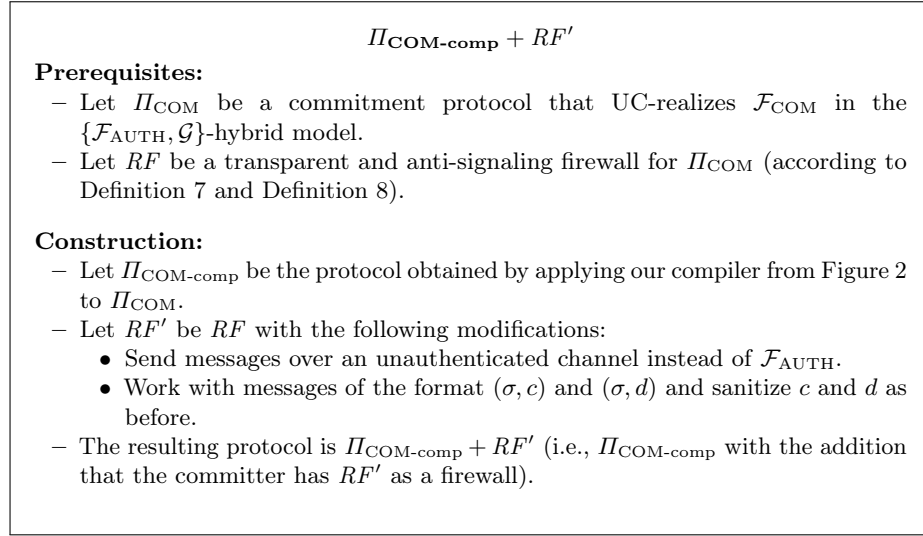


Fig. 6: The concrete construction of our subversion- and impersonation-resilient commitment protocol.

a malicious firewall can still impersonate the honest committer. Then we apply our compiler from Figure 2 to Π_{COM} and add a slightly modified version of RF , which now only communicates over an unauthenticated channel. This gives us a subversion- and impersonation-resilient protocol $\Pi := \Pi_{\text{COM-comp}} + RF'$, in particular, a malicious firewall can no longer impersonate an honest committer. See Lemmas 7 and 8 for the proofs that RF' is transparent and anti-signaling.

For showing that Π^s srUC-realizes $\mathcal{F}_{\text{COM}}^r$, we first show that Π^s srUC-realizes $\Pi_{\text{COM-comp}}^r$, where an honest committer with a malicious firewall in Π^s is mapped to an honest committer in $\Pi_{\text{COM-comp}}^r$. Then, we use the fact that $\Pi_{\text{COM-comp}}$ UC-realizes $\mathcal{F}_{\text{COM}}$ (by Theorem 1) and apply Lemma 5 to get that $\Pi_{\text{COM-comp}}^r \geq_{\text{srUC}} \mathcal{F}_{\text{COM}}^r$. Now we can take the previous two steps and apply srUC transitivity (Lemma 6) to get that Π^s srUC-realizes $\mathcal{F}_{\text{COM}}^r$. Subversion resilience is reflected by the fact that we show srUC-realization of $\mathcal{F}_{\text{COM}}^r$. Resilience to

impersonation by a firewall is reflected by the fact that an honest committer with a malicious firewall in Π^s is mapped to an honest committer in $\mathcal{F}_{\text{COM}}^r$.

Theorem 3 (Subversion Resilience of $\Pi_{\text{COM-comp}}$ with RF'). *Let Π_{COM} be a non-interactive commitment protocol (Definition 10) that UC-realizes $\mathcal{F}_{\text{COM}}$ in the $\{\mathcal{F}_{\text{AUTH}}, \mathcal{G}\}$ -hybrid model. Let RF be a strongly transparent and anti-signaling firewall for Π_{COM} . Let $\Pi_{\text{COM-comp}}$ and RF' be as described in Figure 6. Let $\Pi := \Pi_{\text{COM-comp}} + \text{RF}'$. Then $\Pi^s \geq_{\text{srUC}} \Pi_{\text{COM-comp}}^r$ in the $\{\mathcal{F}_{\text{KRK}}, \mathcal{G}\}$ -hybrid model, where an honest committer with a malicious firewall in Π is treated as an honest committer in $\Pi_{\text{COM-comp}}^r$.*

The theorem's statement essentially says that Π^s is subversion and impersonation resilient. The resilience to subversion attacks is covered by the fact that Π^s allows subversion attacks and is at least as secure as $\Pi_{\text{COM-comp}}^r$, where subversion attacks are ignored. The resilience to impersonation from the firewall is covered by the fact that an honest committer with a malicious firewall in Π^s is treated as an honest committer in $\Pi_{\text{COM-comp}}^r$.

Note that here, the simulator does not live in the ideal world, but instead in a version of the real world where it interacts with the real protocol $\Pi_{\text{COM-comp}}^r$ and needs to simulate Π^s . Thus, the main task of the simulator is to handle the absence of the firewall. The proof for this turns out to be quite simple as we prove resilience to subversion attacks and impersonation within the real world, following [1]. It is only in the next step that we then apply Theorem 1 and transitivity to get to the ideal world. This modular analysis is possible through careful alignment of all required building blocks. We defer the proof to Appendix E.2.

Now we have everything in place to show that $\Pi_{\text{COM-comp}}$ with RF' is a secure, subversion and impersonation resilient commitment protocol. For the proof we only need to chain together the previous theorem and lemmas.

Corollary 1. *Let Π_{COM} be a non-interactive commitment (Definition 10) protocol that UC-realizes $\mathcal{F}_{\text{COM}}$ in the $\{\mathcal{F}_{\text{AUTH}}, \mathcal{G}\}$ -hybrid model and RF a strongly transparent and anti-signaling firewall for Π_{COM} . Let $\Pi_{\text{COM-comp}}$ and RF' be as described in Figure 6. Let $\Pi := \Pi_{\text{COM-comp}} + \text{RF}'$. Then $\Pi^s \geq_{\text{srUC}} \mathcal{F}_{\text{COM}}^r$ in the $\{\mathcal{F}_{\text{KRK}}, \mathcal{G}\}$ -hybrid model, where an honest committer with a malicious firewall in Π is treated as an honest committer in $\mathcal{F}_{\text{COM}}^r$.*

Proof. By Theorem 3, we get $\Pi^s \geq_{\text{srUC}} \Pi_{\text{COM-comp}}^r$. By Theorem 1, we get that $\Pi_{\text{COM-comp}}^r \geq_{\text{UC}} \mathcal{F}_{\text{COM}}$ and by applying Lemma 5, we get $\Pi_{\text{COM-comp}}^r \geq_{\text{srUC}} \mathcal{F}_{\text{COM}}^r$. Then, by applying srUC transitivity (Lemma 6), we get $\Pi^s \geq_{\text{srUC}} \mathcal{F}_{\text{COM}}^r$.

Security in the *Original* ABMO Model and the *New* CMNV Model

The statements from Theorem 3 and Corollary 1 also hold in the *original* ABMO model (i.e., the model from [1] without our modifications from Section 4). The only relevant difference is that in the *original* ABMO model, subverted codes are restricted to modify a party's randomness. Thus, we only need to reconsider the corruption case of a subverted committer with an honest firewall. This case

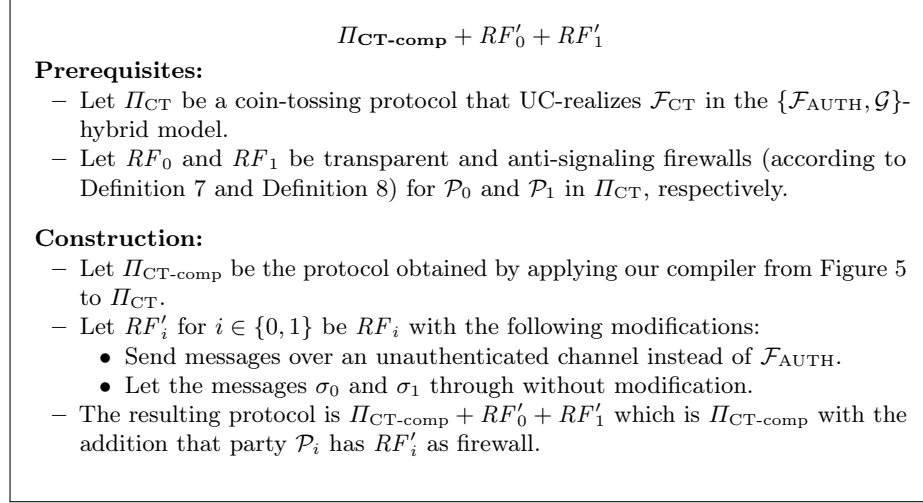


Fig. 7: The concrete construction of our subversion- and impersonation-resilient coin-tossing protocol.

resolves to an honest committer as the firewall sanitizes the messages of Π_{COM} , and the signatures do not need any randomness (as they are unique signatures). The new CMNV model is almost identical to the new ABMO model, so the proofs also apply there. Protocols in the original CMNV model typically use the $\mathcal{F}_{\text{SAT}}$ functionality for communication. We can also apply our techniques from Section 3 if a protocol uses $\mathcal{F}_{\text{SAT}}$ instead of $\mathcal{F}_{\text{AUTH}}$, as long as the protocol realizes the standard commitment functionality and the firewall is transparent except for its use of $\mathcal{F}_{\text{SAT}}$.

5.2 Coin Toss

We now turn towards presenting a subversion- and impersonation-resilient coin-tossing protocol, with the construction of it being presented in Figure 7. The construction and proof is almost analogous to the case of the commitment protocol in Section 5.1.

Lemma 1. *If RF_i ($i \in \{0, 1\}$) is a strongly transparent firewall (see Definition 7) for Π_{CT} , then RF'_i is a strongly transparent firewall for $\Pi_{\text{CT-comp}}$ (as defined in Figure 7).*

Proof (sketch). The argument is essentially the same as for Lemma 7.

Lemma 2. *If RF_i ($i \in \{0, 1\}$) is an anti-signaling firewall (see Definition 8) for Π_{CT} , then RF'_i is an anti-signaling firewall for $\Pi_{\text{CT-comp}}$ (as defined in Figure 7).*

Proof (sketch). The argument is almost the same as for Lemma 8. Because RF_i is an anti-signaling firewall for Π_{CT} , the messages of Π_{CT} are already sanitized. Also, because σ_i is a unique signature, a speciously subverted party cannot modify them without being detected.

Theorem 4 (Subversion Resilience of $\Pi_{CT\text{-comp}}$ with RF'_0 and RF'_1). *Let Π_{CT} be a protocol that UC-realizes $\mathcal{F}_{CT}$ in the $\{\mathcal{F}_{AUTH}, \mathcal{G}\}$ -hybrid model. Let RF_i ($i \in \{0, 1\}$) be strongly transparent and anti-signaling firewalls for Π_{CT} . Let $\Pi_{CT\text{-comp}}$ and RF'_i be as described in Figure 7. Let $\Pi := \Pi_{CT\text{-comp}} + RF'_0 + RF'_1$. Then $\Pi^s \geq_{srUC} \Pi_{CT\text{-comp}}^r$ in the $\{\mathcal{F}_{KRK}, \mathcal{G}\}$ -hybrid model, where an honest party with a malicious firewall in Π is treated as an honest party in $\Pi_{CT\text{-comp}}^r$.*

The proof is very similar to the proof of Theorem 3 as the main task of the simulator is to simulate the effect of the firewall. The proof can be found in Appendix E.3.

Analogously to the case of our commitment protocol, we can now show that $\Pi_{CT\text{-comp}}$ with RF' is a secure, subversion and impersonation resilient coin-toss protocol. The proof is analogous to the proof of our commitment protocol.

Corollary 2. *Let Π_{CT} be a protocol that UC-realizes $\mathcal{F}_{CT}$ in the $\{\mathcal{F}_{AUTH}, \mathcal{G}\}$ -hybrid model. Let RF_i ($i \in \{0, 1\}$) be strongly transparent and anti-signaling firewalls for Π_{CT} . Let $\Pi_{CT\text{-comp}}$ and RF'_i be as described in Figure 7. Let $\Pi := \Pi_{CT\text{-comp}} + RF'_0 + RF'_1$. Then $\Pi^s \geq_{srUC} \mathcal{F}_{CT}^r$ in the $\{\mathcal{F}_{KRK}, \mathcal{G}\}$ -hybrid model, where an honest party with a malicious firewall in Π is treated as an honest party in $\mathcal{F}_{CT}^r$.*

Proof (sketch). The proof is analogous to the proof of Corollary 1.

Acknowledgement. We thank the anonymous reviewers from Asiacrypt 2026 for their valuable feedback. The authors also acknowledge the use of AI tools (Claude) to review a draft of this paper for typos and inconsistencies. All resulting changes were made and verified by the authors. This work was supported by funding by the German Federal Ministry of Research, Technology and Space (BMFTR) under the project “Sec4IoMT” (ID 16KIS1692 and 16KIS1693), by the European Union’s Horizon Europe programme (FOCAL project, Grant Agreement No. 101225859), by the Topic Engineering Secure Systems of the Helmholtz Association (HGF), and by KASTEL Security Research Labs, Karlsruhe.

References

1. Arnold, P., Berndt, S., Müller-Quade, J., and Ottenhues, A.: Protection Against Subversion Corruptions via Reverse Firewalls in the Plain Universal Composability Framework. In: Fischlin, M., and Moonsamy, V. (eds.) ACNS 2025, Part II. LNCS, pp. 510–539. Springer, Cham (2025) (cit. on pp. 4–10, 18–20, 24, 25, 27, 36). DOI: [10.1007/978-3-031-95764-2_20](https://doi.org/10.1007/978-3-031-95764-2_20)

2. Arnold, P., Berndt, S., Müller-Quade, J., and Ottenhues, A.: Protection Against Subversion Corruptions via Reverse Firewalls in the Plain Universal Composability Framework. *Cryptology ePrint Archive*, Paper 2023/1951 (2023) (cit. on pp. 4, 7, 14)
3. Ateniese, G., Magri, B., and Venturi, D.: Subversion-Resilient Signature Schemes. In: Ray, I., Li, N., and Kruegel, C. (eds.) *ACM CCS 2015*, pp. 364–375. ACM Press (2015) (cit. on pp. 6, 8, 21). DOI: [10.1145/2810103.2813635](https://doi.org/10.1145/2810103.2813635)
4. Bellare, M., Jaeger, J., and Kane, D.: Mass-surveillance without the State: Strongly Undetectable Algorithm-Substitution Attacks. In: Ray, I., Li, N., and Kruegel, C. (eds.) *ACM CCS 2015*, pp. 1431–1440. ACM Press (2015) (cit. on pp. 2, 21). DOI: [10.1145/2810103.2813681](https://doi.org/10.1145/2810103.2813681)
5. Bellare, M., Paterson, K.G., and Rogaway, P.: Security of Symmetric Encryption against Mass Surveillance. In: Garay, J.A., and Gennaro, R. (eds.) *CRYPTO 2014, Part I. LNCS*, pp. 1–19. Springer, Berlin, Heidelberg (2014) (cit. on pp. 2, 6, 21). DOI: [10.1007/978-3-662-44371-2_1](https://doi.org/10.1007/978-3-662-44371-2_1)
6. Bemmman, P., Berndt, S., and Chen, R.: Subversion-Resilient Signatures Without Random Oracles. In: Pöpper, C., and Batina, L. (eds.) *ACNS 2024, Part I. LNCS*, pp. 351–375. Springer, Cham (2024) (cit. on p. 6). DOI: [10.1007/978-3-031-54770-6_14](https://doi.org/10.1007/978-3-031-54770-6_14)
7. Bemmman, P., Berndt, S., Diemert, D., Eisenbarth, T., and Jager, T.: Subversion-Resilient Authenticated Encryption Without Random Oracles. In: Tibouchi, M., and Wang, X. (eds.) *ACNS 2023, Part II. LNCS*, pp. 460–483. Springer, Cham (2023) (cit. on p. 6). DOI: [10.1007/978-3-031-33491-7_17](https://doi.org/10.1007/978-3-031-33491-7_17)
8. Bemmman, P., Chen, R., and Jager, T.: Subversion-Resilient Public Key Encryption with Practical Watchdogs. In: Garay, J. (ed.) *PKC 2021, Part I. LNCS*, pp. 627–658. Springer, Cham (2021) (cit. on p. 6). DOI: [10.1007/978-3-030-75245-3_23](https://doi.org/10.1007/978-3-030-75245-3_23)
9. Berndt, S., and Liskiewicz, M.: Algorithm Substitution Attacks from a Steganographic Perspective. In: Thuraisingham, B.M., Evans, D., Malkin, T., and Xu, D. (eds.) *ACM CCS 2017*, pp. 1649–1660. ACM Press (2017) (cit. on pp. 2, 21). DOI: [10.1145/3133956.3133981](https://doi.org/10.1145/3133956.3133981)
10. Beskorovajnov, W., Gröll, R., Müller-Quade, J., Ottenhues, A., and Schwerdt, R.: A New Security Notion for PKC in the Standard Model: Weaker, Simpler, and Still Realizing Secure Channels. In: Hanaoka, G., Shikata, J., and Watanabe, Y. (eds.) *PKC 2022, Part II. LNCS*, pp. 316–344. Springer, Cham (2022) (cit. on p. 35). DOI: [10.1007/978-3-030-97131-1_11](https://doi.org/10.1007/978-3-030-97131-1_11)
11. Blum, M.: Coin Flipping by Telephone. In: *Proc. IEEE Spring COMPCOM*, pp. 133–137 (1982) (cit. on p. 38)
12. Camenisch, J., Drijvers, M., and Lehmann, A.: Anonymous Attestation with Subverted TPMs. In: Katz, J., and Shacham, H. (eds.) *CRYPTO 2017, Part III. LNCS*, pp. 427–461. Springer, Cham (2017) (cit. on p. 6). DOI: [10.1007/978-3-319-63697-9_15](https://doi.org/10.1007/978-3-319-63697-9_15)
13. Canetti, R.: Universally Composable Security. *J. ACM* 67(5), 28:1–28:94 (2020) (cit. on pp. 8, 11, 34). DOI: [10.1145/3402457](https://doi.org/10.1145/3402457)
14. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. *IACR Cryptol. ePrint Arch.* (2000) (cit. on pp. 11, 49). DOI: [10.1109/SFCS.2001.959888](https://doi.org/10.1109/SFCS.2001.959888)
15. Canetti, R., and Fischlin, M.: Universally Composable Commitments. In: Kilian, J. (ed.) *CRYPTO 2001. LNCS*, pp. 19–40. Springer, Berlin, Heidelberg (2001) (cit. on p. 35). DOI: [10.1007/3-540-44647-8_2](https://doi.org/10.1007/3-540-44647-8_2)

16. Chakraborty, S., Dziembowski, S., and Nielsen, J.B.: Reverse Firewalls for Actively Secure MPCs. In: Micciancio, D., and Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, pp. 732–762. Springer, Cham (2020) (cit. on p. 6). DOI: [10.1007/978-3-030-56880-1_26](https://doi.org/10.1007/978-3-030-56880-1_26)
17. Chakraborty, S., Ganesh, C., Pancholi, M., and Sarkar, P.: Reverse Firewalls for Adaptively Secure MPC Without Setup. In: Tibouchi, M., and Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, pp. 335–364. Springer, Cham (2021) (cit. on p. 6). DOI: [10.1007/978-3-030-92075-3_12](https://doi.org/10.1007/978-3-030-92075-3_12)
18. Chakraborty, S., Ganesh, C., and Sarkar, P.: Reverse Firewalls for Oblivious Transfer Extension and Applications to Zero-Knowledge. In: Hazay, C., and Stam, M. (eds.) EUROCRYPT 2023, Part I. LNCS, pp. 239–270. Springer, Cham (2023) (cit. on p. 6). DOI: [10.1007/978-3-031-30545-0_9](https://doi.org/10.1007/978-3-031-30545-0_9)
19. Chakraborty, S., Magliocco, L., Magri, B., and Venturi, D.: Key Exchange in the Post-snowden Era: Universally Composable Subversion-Resilient PAKE. In: Chung, K.-M., and Sasaki, Y. (eds.) ASIACRYPT 2024, Part V. LNCS, pp. 101–133. Springer, Singapore (2024) (cit. on pp. 5, 7, 8, 39). DOI: [10.1007/978-981-96-0935-2_4](https://doi.org/10.1007/978-981-96-0935-2_4)
20. Chakraborty, S., Magri, B., Nielsen, J.B., and Venturi, D.: Universally Composable Subversion-Resilient Cryptography. In: Dunkelman, O., and Dziembowski, S. (eds.) EUROCRYPT 2022, Part I. LNCS, pp. 272–302. Springer, Cham (2022) (cit. on pp. 4–9, 18, 19, 21, 22, 24, 25). DOI: [10.1007/978-3-031-06944-4_10](https://doi.org/10.1007/978-3-031-06944-4_10)
21. Checkoway, S., Maskiewicz, J., Garman, C., Fried, J., Cohnsey, S., Green, M., Heninger, N., Weinmann, R.-P., Rescorla, E., and Shacham, H.: A Systematic Analysis of the Juniper Dual EC Incident. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., and Halevi, S. (eds.) ACM CCS 2016, pp. 468–479. ACM Press (2016) (cit. on p. 2). DOI: [10.1145/2976749.2978395](https://doi.org/10.1145/2976749.2978395)
22. Checkoway, S., Niederhagen, R., Everspaugh, A., Green, M., Lange, T., Ristenpart, T., Bernstein, D.J., Maskiewicz, J., Shacham, H., and Fredrikson, M.: On the Practical Exploitability of Dual EC in TLS Implementations. In: Fu, K., and Jung, J. (eds.) USENIX Security 2014, pp. 319–335. USENIX Association (2014) (cit. on p. 2)
23. Chow, S.S.M., Russell, A., Tang, Q., Yung, M., Zhao, Y., and Zhou, H.-S.: Let a Non-barking Watchdog Bite: Cliptographic Signatures with an Offline Watchdog. In: Lin, D., and Sako, K. (eds.) PKC 2019, Part I. LNCS, pp. 221–251. Springer, Cham (2019) (cit. on p. 6). DOI: [10.1007/978-3-030-17253-4_8](https://doi.org/10.1007/978-3-030-17253-4_8)
24. Discussion about Kyber’s tweaked FO transform, Discussion Thread on the PQC mailing list. (2023). <https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/WFRDl8DqYQ4> (cit. on p. 2)
25. Dodis, Y., Magri, B., Stephens-Davidowitz, N., and Tselekounis, Y.: Guarding the Signal: Secure Messaging with Reverse Firewalls. In: Kalai, Y.T., and Kamara, S.F. (eds.) CRYPTO 2025, Part VIII. LNCS, pp. 39–74. Springer, Cham (2025) (cit. on p. 8). DOI: [10.1007/978-3-032-01913-4_2](https://doi.org/10.1007/978-3-032-01913-4_2)
26. Dodis, Y., Mironov, I., and Stephens-Davidowitz, N.: Message Transmission with Reverse Firewalls—Secure Communication on Corrupted Machines. In: Robshaw, M., and Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, pp. 341–372. Springer, Berlin, Heidelberg (2016) (cit. on p. 6). DOI: [10.1007/978-3-662-53018-4_13](https://doi.org/10.1007/978-3-662-53018-4_13)
27. Ganesh, C., Magri, B., and Venturi, D.: Cryptographic Reverse Firewalls for Interactive Proof Systems. In: Czumaj, A., Dawar, A., and Merelli, E. (eds.) ICALP 2020. LIPIcs, 55:1–55:16. Schloss Dagstuhl (2020) (cit. on p. 6). DOI: [10.4230/LIPIcs.ICALP.2020.55](https://doi.org/10.4230/LIPIcs.ICALP.2020.55)

28. Hofheinz, D., Müller-Quade, J., and Unruh, D.: On the (Im-)Possibility of Extending Coin Toss. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, pp. 504–521. Springer, Berlin, Heidelberg (2006) (cit. on p. 36). doi: [10.1007/11761679_30](https://doi.org/10.1007/11761679_30)
29. Kumar, C., Mishra, N., Chakraborty, S., Ghosh, S., and Mukhopadhyay, D.: TERRA: Trojan-Resilient Reverse-Firewall for Cryptographic Applications. In: Nicomette, V., Benzekri, A., Boulahia-Cuppens, N., and Vaidya, J. (eds.) ESORICS 2025, Part II. LNCS, pp. 226–246. Springer, Cham (2025) (cit. on pp. 7, 8). doi: [10.1007/978-3-032-07891-9_12](https://doi.org/10.1007/978-3-032-07891-9_12)
30. Li, G., Liu, J., Zhang, Z., and 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) (cit. on p. 6). doi: [10.1007/978-3-030-88323-2_5](https://doi.org/10.1007/978-3-030-88323-2_5)
31. Liu, J., Chen, R., Wang, Y., Tang, X., and Su, J.: Subversion-Resilient Authenticated Key Exchange with Reverse Firewalls. In: Liu, J.K., Chen, L., Sun, S.-F., and Liu, X. (eds.) ProvSec 2023, Part II. LNCS, pp. 181–200. Springer, Singapore (2024) (cit. on p. 8). doi: [10.1007/978-981-96-0957-4_10](https://doi.org/10.1007/978-981-96-0957-4_10)
32. Liu, J., Wang, Y., Chen, R., Huang, X., Su, J., and Yung, M.: Universally Composable Subversion-Resilient Authenticated Key Exchange. In: Hanaoka, G., and Yang, B.-Y. (eds.) ASIACRYPT 2025, Part II. LNCS, pp. 103–136. Springer, Singapore (2025) (cit. on pp. 4, 5, 7, 8, 11). doi: [10.1007/978-981-95-5096-8_4](https://doi.org/10.1007/978-981-95-5096-8_4)
33. Liu, J., Wang, Y., Chen, R., Tang, X., and Su, J.: srCPace: Universally Composable PAKE with Subversion-Resilience. In: Lin, D., Wang, M., and Yung, M. (eds.) Inscrypt. Lecture Notes in Computer Science, pp. 211–231. Springer (2024) (cit. on pp. 5, 7, 8). doi: [10.1007/978-981-96-4731-6_11](https://doi.org/10.1007/978-981-96-4731-6_11)
34. Liu, J., Wang, Y., Tang, X., Chen, R., Huang, X., and Su, J.: srTLS: Secure TLS Handshake on Corrupted Machines. IEEE Trans. Dependable Secur. Comput. 22(3), 3137–3154 (2025) (cit. on p. 5). doi: [10.1109/TDSC.2024.3524681](https://doi.org/10.1109/TDSC.2024.3524681)
35. Lysyanskaya, A.: Unique Signatures and Verifiable Random Functions from the DH-DDH Separation. In: Yung, M. (ed.) CRYPTO 2002. LNCS, pp. 597–612. Springer, Berlin, Heidelberg (2002) (cit. on p. 37). doi: [10.1007/3-540-45708-9_38](https://doi.org/10.1007/3-540-45708-9_38)
36. Mironov, I., and Stephens-Davidowitz, N.: Cryptographic Reverse Firewalls. In: Oswald, E., and Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, pp. 657–686. Springer, Berlin, Heidelberg (2015) (cit. on pp. 2, 6, 7). doi: [10.1007/978-3-662-46803-6_22](https://doi.org/10.1007/978-3-662-46803-6_22)
37. Russell, A., Tang, Q., Yung, M., and Zhou, H.-S.: Cliptography: Clipping the Power of Kleptographic Attacks. In: Cheon, J.H., and Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, pp. 34–64. Springer, Berlin, Heidelberg (2016) (cit. on pp. 2, 6). doi: [10.1007/978-3-662-53890-6_2](https://doi.org/10.1007/978-3-662-53890-6_2)
38. Russell, A., Tang, Q., Yung, M., and Zhou, H.-S.: Correcting Subverted Random Oracles. In: Shacham, H., and Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, pp. 241–271. Springer, Cham (2018) (cit. on p. 6). doi: [10.1007/978-3-319-96881-0_9](https://doi.org/10.1007/978-3-319-96881-0_9)
39. Russell, A., Tang, Q., Yung, M., and Zhou, H.-S.: Generic Semantic Security against a Kleptographic Adversary. In: Thuraisingham, B.M., Evans, D., Malkin, T., and Xu, D. (eds.) ACM CCS 2017, pp. 907–922. ACM Press (2017) (cit. on p. 6). doi: [10.1145/3133956.3133993](https://doi.org/10.1145/3133956.3133993)
40. Young, A., and Yung, M.: Kleptography: Using Cryptography Against Cryptography. In: Fumy, W. (ed.) EUROCRYPT’97. LNCS, pp. 62–74. Springer, Berlin, Heidelberg (1997) (cit. on pp. 2, 21). doi: [10.1007/3-540-69053-0_6](https://doi.org/10.1007/3-540-69053-0_6)

41. Young, A., and Yung, M.: The Dark Side of “Black-Box” Cryptography, or: Should We Trust Capstone? In: Kobitz, N. (ed.) CRYPTO’96. LNCS, pp. 89–103. Springer, Berlin, Heidelberg (1996) (cit. on p. 2). DOI: [10.1007/3-540-68697-5_8](https://doi.org/10.1007/3-540-68697-5_8)

A Preliminaries (Continued)

A.1 Glossary

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

$\mathcal{A}$	Adversary
$\mathcal{C}$	Real code running by a party (or its body)
$\tilde{\mathcal{C}}$	Specious code running by a party (or its body)
$\mathcal{F}$	Ideal functionality
κ	Security parameter
$\mathcal{P}$	Party
$\hat{\mathcal{P}}$	Incorruptible party, a party that ignores all corruptions
$\bar{\mathcal{P}}$	Subvertable party $\mathcal{P}$
Π	Protocol
PPT	Probabilistic polynomial time, i.e., efficient randomized algorithms
RF	Cryptographic reverse firewall, which is itself also a party
$\hat{RF}$	Incorruptible firewall, a firewall that ignores all corruptions
$RF \circ \mathcal{P}$	Composed party consisting of a party $\mathcal{P}$ and a firewall RF
$(\dots)^r$	$\Pi/\mathcal{F}$ with dummy <i>reveal secret</i> interface and which ignore subversion corruptions
$(\dots)^s$	Protocols which accept subversion corruptions
Sim	Simulator
$\mathcal{Z}$	Environment
$A \geq_{UC} B$	A UC-realizes B
$A \geq_{srUC} B$	A subversion-resilient-UC-realizes B
$A \approx B$	A is computational indistinguishable from B

A.2 Ideal UC Functionalities

In the following, we will explain some ideal functionalities which will be used throughout this paper.

Definition 2 (The Ideal Functionality for unauthenticated channels).

$\mathcal{F}_{unAUTH}$
– Upon invocation, with input $(\text{send}, sid, \mathcal{R}, m)$ from S, send backdoor message $(\text{send}, sid, S, \mathcal{R}, m)$ to the adversary. – Upon receiving backdoor message $(\text{sendOK}, sid, S, \mathcal{R}, m')$ from adversary, send (sendOK, sid, m') to $\mathcal{R}$ in the name of S. – Ignore all other inputs and backdoor messages.

Definition 3 (The Ideal Functionality $\mathcal{F}_{AUTH}$, adapted from [13]).

$\mathcal{F}_{AUTH}$

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

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

Definition 4 (The Ideal Functionality $\mathcal{F}_{KRK}$, adapted from [10]).

 $\mathcal{F}_{KRK}$

The ideal functionality key registration with knowledge of a public key infrastructure which provides for each party $\mathcal{P}_i$ a key pair consisting of public key pk_i and secret key sk_i whereby any party $\mathcal{P}_j$ only receives the public key of another party.

Registration:

- Upon receiving $(\text{register}, \text{sid})$ from a party $\mathcal{P}_i$:
 1. check that $(\text{sid}, \mathcal{P}_i, \cdot, \cdot)$ is not recorded,
 2. generate (sk_i, pk_i) , and
 3. send $(\text{registered}, \text{sid}, \mathcal{P}_i, pk_i)$ to the adversary $\mathcal{A}$.
- Upon receiving $(\text{register ok}, \text{sid})$ from $\mathcal{A}$:
 1. record $(\text{sid}, \mathcal{P}_i, sk_i, pk_i)$, and
 2. send $(\text{registered}, \text{sid}, sk_i, pk_i)$ to $\mathcal{P}_i$.
- Ignore further messages $(\text{register}, \text{sid})$ from $\mathcal{P}_i$.

Retrieving:

- Upon receiving $(\text{retrieve}, \text{sid}, \mathcal{P}_i)$ from a party $\mathcal{P}_j$ or $\mathcal{A}$:
 1. retrieve $(\text{sid}, \mathcal{P}_i, \cdot, pk_i)$,
if record does not exists set $pk_i := \perp$, and
 2. send $(\text{retrieved}, \text{sid}, \mathcal{P}_i, pk_i)$ to $\mathcal{P}_j$ or $\mathcal{A}$, respectively.

Definition 5 (The Ideal Functionality $\mathcal{F}_{COM}$, adapted from [15]).

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

The ideal commitment functionality, $\mathcal{F}_{COM}$, between committer $\mathcal{C}$ and verifier $\mathcal{V}$.

- Upon receiving an input $(\text{commit}, \text{sid}, m)$ from $\mathcal{C}$, where $m \in \{0, 1\}^*$, record the tuple (sid, m) and generate a public delayed output $(\text{committed}, \text{sid})$ to $\mathcal{V}$. Ignore subsequent commands of the form $(\text{commit}, \text{sid}, \cdot)$.
- Upon receiving an input $(\text{open}, \text{sid})$ from $\mathcal{C}$, generate a public delayed output $(\text{opened}, \text{sid}, m)$ to $\mathcal{V}$. Ignore subsequent commands of the form $(\text{open}, \text{sid}, \cdot)$.
- Ignore all further inputs.

Definition 6 (The Ideal Functionality $\mathcal{F}_{CT}$, adapted from [28]).

$$\mathcal{F}_{CT}$$

The ideal coin-tossing functionality, $\mathcal{F}_{CT}$, between two parties $\mathcal{P}_0$ and $\mathcal{P}_1$.

- Upon receiving an input $(\text{toss}, \text{sid})$ from $\mathcal{P}_i$: If this is the first message of this kind from $\mathcal{P}_i$, record the tuple $(\text{sid}, \mathcal{P}_i)$. If the record for $\mathcal{P}_{1-i}$ does not exist, send $(\text{received}, \text{sid}, \mathcal{P}_i)$ to $\mathcal{A}$. Else, (if the record for $\mathcal{P}_{1-i}$ does exist), sample a uniformly distributed $r \in \{0, 1\}^\kappa$ and send (sid, r) to $\mathcal{A}$.
- Upon receiving an input $(\text{deliver}, \text{sid}, \mathcal{P}_i)$ from $\mathcal{A}$ and r has already been sampled, send $(\text{tossed}, \text{sid}, r)$ to $\mathcal{P}_i$. Ignore subsequent commands of the form $(\text{deliver}, \text{sid}, \mathcal{P}_i)$.
- Ignore all further inputs.

A.3 Properties of Reverse Firewalls

The following are the most important properties of such firewalls for this work, where Definitions 7 and 8 are taken and minorly adapted from [1]. One difference is that we added the restriction of $\mathcal{Z}$ to be *specious* in Definition 8.

Definition 7 (Transparency of Firewalls). Let $\mathcal{P}$ be a party, RF its firewall, and $\hat{\mathcal{P}}$ (resp. $\hat{RF}$) be the incorruptible versions of $\mathcal{P}$ (resp. RF). Furthermore, let Π be the protocol run by $\hat{\mathcal{P}}$, and Π' be the protocol run by the composed party $\hat{RF} \circ \hat{\mathcal{P}}$. The firewall RF provides outer transparency if for all PPT environments $\mathcal{Z}$, it holds that $|\Pr[\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, r) = 1] - \Pr[\text{EXEC}_{\Pi', \mathcal{A}, \mathcal{Z}}(\kappa, r) = 1]|$ is negligible, where $\mathcal{A}$ is the dummy adversary. If the protocols Π and Π' contain the same code $\mathcal{C}$ for their respective party $\hat{\mathcal{P}}$, then RF provides inner transparency. If RF provides both inner and outer transparency, then it provides strong transparency.

Definition 8 (Anti-Signaling Firewalls). Let $\mathcal{P}$ be a party and RF its firewall. Let $\overline{\mathcal{P}}$ be the subvertable version of $\mathcal{P}$. Let $\hat{\mathcal{P}}$ and $\hat{RF}$ be the incorruptible versions of $\mathcal{P}$ and RF , respectively. Let Π be the protocol run by the composed party $\hat{RF} \circ \overline{\mathcal{P}}$. Let Π' be the protocol run by $\hat{\mathcal{P}}$ and $\hat{RF}$. The firewall RF is anti-signaling if for all specious PPT environments $\mathcal{Z}$, it holds that $|\Pr[EXEC_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, r) = 1] - \Pr[EXEC_{\Pi', \mathcal{A}, \mathcal{Z}}(\kappa, r) = 1]|$ is negligible, where $\mathcal{A}$ is the dummy adversary.

A.4 Cryptographic Primitives (Continued)

The most relevant cryptographic primitives of this work are unique signatures, non-interactive UC-secure commitments and coin toss.

Unique Signature Scheme A signature scheme $(\text{Kgen}, \text{Sign}, \text{Vfy})$ is called *unique* if the following definition holds. The defining property of a unique signature scheme is that for each message m , there is only *one* valid signature.

Definition 9 (Unique Signature Scheme adapted from [35]). A function family $\sigma(\cdot): \{0, 1\}^* \rightarrow \{0, 1\}^{\ell(\kappa)}$ is a unique signature scheme $USig$ if there exists a probabilistic algorithm $USig.\text{Kgen}$, an efficient deterministic algorithm $USig.\text{Sign}$, and a probabilistic algorithm $USig.\text{Vfy}$ such that: $USig.\text{Kgen}(1^\kappa)$ generates the key pair (pk, sk) , $USig.\text{Sign}(sk, m)$ computes the value $\sigma(m)$, and $USig.\text{Vfy}(pk, m, \bar{\sigma})$ verifies that $\bar{\sigma} = \sigma(m)$. More formally (but assuming, for simplicity, that $USig.\text{Vfy}$ is a deterministic algorithm¹³): We have **Uniqueness of $\sigma(m)$** , if there do not exist values $(pk, m, \sigma_1, \sigma_2)$ such that $\sigma_1 \neq \sigma_2$ and $USig.\text{Vfy}(pk, m, \sigma_1) = USig.\text{Vfy}(pk, m, \sigma_2) = 1$. Security is given by the fact that for all families of PPT oracle Turing machines $\{\mathcal{A}_k^{(\cdot)}\}$, there exists a negligible function $\nu(\kappa)$ such that

$$\Pr \left[(pk, sk) \leftarrow USig.\text{Kgen}(1^\kappa); (m, \sigma) \leftarrow \mathcal{A}_k^{USig.\text{Sign}(sk, \cdot)}(1^\kappa) : \right. \\ \left. USig.\text{Vfy}(pk, m, \sigma) = 1 \wedge m \notin Q \right] \leq \nu(\kappa),$$

where Q is the set of messages that $\mathcal{A}_k$ queried to its signing oracle.

Non-interactive UC-secure commitments A standard commitment scheme consists of a single PPT algorithm 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 $\text{Com}(v, r) = \text{Com}(v', r')$. For Universal Composability, the commitment additionally fulfills the properties of *extractability* and *equivocality*. The ideal functionality for a UC commitment is given in Appendix A.2. In this paper, we make use

¹³ In case this is not true, the adjustment to the definition is straightforward.

of UC-secure non-interactive commitments, meaning that communication goes only from committer $\mathcal{C}$ to verifier $\mathcal{V}$. However, we will need to directly access the messages sent by the committer and we also need to directly give messages to the verifier. The standard UC formalization of a commitment does not directly allow this. Therefore, we write both the committer and the verifier as a tuple of algorithms $(\text{COM}^{\mathcal{C}}.\text{Commit}, \text{COM}^{\mathcal{C}}.\text{Open})$ and $(\text{COM}^{\mathcal{V}}.\text{RcvCom}, \text{COM}^{\mathcal{V}}.\text{Vfy})$. To rewrite a non-interactive UC-secure commitment protocol with a committer $\mathcal{C}$ and a verifier $\mathcal{V}$ into our notation, we define the scheme as follows. We assume that the commitment protocol uses $\mathcal{F}_{\text{AUTH}}$ for communication. Our approach, of course, also works with any other standard communication functionality.

Definition 10 (Non-interactive UC-secure commitments).

- $\text{COM}^{\mathcal{C}}.\text{Commit}(sid, m)$: Call $\mathcal{C}$'s commit interface on (sid, m) and output the message α that $\mathcal{C}$ sends to $\mathcal{F}_{\text{AUTH}}$.
- $\text{COM}^{\mathcal{C}}.\text{Open}(sid)$: Call $\mathcal{C}$'s open interface on (sid) and output the message β that $\mathcal{C}$ sends to $\mathcal{F}_{\text{AUTH}}$.
- $\text{COM}^{\mathcal{V}}.\text{RcvCom}(sid, \alpha)$: Give α to $\mathcal{V}$ as message from $\mathcal{F}_{\text{AUTH}}$ with sender $\mathcal{C}$ and run $\mathcal{V}$'s code from there. Output the value that $\mathcal{V}$ outputs. If $\mathcal{V}$ ends without outputting anything, then output $\perp$.
- $\text{COM}^{\mathcal{V}}.\text{Vfy}(sid, \beta)$: Give β to $\mathcal{V}$ as message from $\mathcal{F}_{\text{AUTH}}$ with sender $\mathcal{C}$ and run $\mathcal{V}$'s code from there. Output the value that $\mathcal{V}$ outputs. If $\mathcal{V}$ ends without outputting anything, then output $\perp$.

Coin Toss A coin toss is an interactive protocol between two parties $\mathcal{P}_0$ and $\mathcal{P}_1$ to generate commonly shared random coins $r \leftarrow \{0, 1\}^{\kappa}$. In the ideal world, both parties inform the ideal functionality, which distributes the coin faithfully. The ideal UC functionality $\mathcal{F}_{\text{CT}}$ is provided in Appendix A.2. Realizations of a coin toss use, e.g., a commitment protocol [11].

B Commitments without Authenticated Channel (Continued)

B.1 Design Choices and Discussion (Continued)

When to compute σ_2 : The signature $\sigma_2 := \text{USig.Sign}(sk, (sid, m))$ in our protocol is computed during the commit phase, but only sent in the opening phase. Instead, the committer could equally well defer the computation of σ_2 to the opening phase. In our formal analysis this does not make any difference.

In practice, an advantage of computing σ_2 in the commit phase is that the signing key does not need to be accessed during the opening phase, which is advantageous if the key is stored on a signature card. An advantage of computing σ_2 during the opening phase is the reduced risk of leaking information about m via timing side channels. Since hiding the message is not a goal of a signature scheme, the sign algorithm may leak information about m through timing side channels. If this happens during the commit phase that is problematic. However, when σ_2 is computed during the opening phase, then the leakage only occurs at the point in time where the committer wants to reveal m anyway.

Sanitizing Signature Public Keys: If we are afraid that the committer is already corrupted at the time they generate and distribute the signature public key, then we need to prevent the attacker from leaking information via that public key. This can be done by letting the firewall rerandomize the public key and signatures. For the pairing based BLS signature scheme, this rerandomization is quite straightforward as described in [19]. In short:

- The sender computes $(pk, sk) := (g^x, x) \leftarrow \text{BLS.Kgen}(1^\kappa)$ and sends $pk = g^x$ to the firewall. The firewall chooses random r and sends $pk' := pk^r = g^{xr}$ to the actual receiver.
- The sender computes a signature $\sigma := H(m)^x$ and sends it to the firewall. The firewall retrieves r and sends $\sigma' := \sigma^r = H(m)^{xr}$ to the actual receiver.
- The receiver can verify the signature by using the standard BLS verification algorithm: $e(pk', H(m)) \stackrel{?}{=} e(g, \sigma')$.

B.2 Proof of Theorem 1

We show that no PPT environment $\mathcal{Z}$ can distinguish between the real world, where it is interacting with the parties and dummy adversary, from the ideal world, where it is interacting with the parties and the simulator. Figure 2 depicts the commitment compiler protocol $\Pi := \Pi_{\text{COM-comp}}$. Figure 3 depicts the simulator $\text{Sim} := \text{Sim}_{\Pi_{\text{COM-comp}}}$. More formally, we show that there exists a simulator Sim for all environments $\mathcal{Z}$ such that

$$\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, r) \approx \text{EXEC}_{\mathcal{F}_{\text{COM}}, \text{Sim}, \mathcal{Z}}(\kappa, r),$$

where κ is the security parameter, r the randomness, and $\mathcal{A}$ the dummy adversary.

We prove Theorem 1 by going through all actions that $\mathcal{Z}$ can take and arguing for each of them that $\mathcal{Z}$ cannot distinguish between real and ideal world. For each of $\mathcal{Z}$'s actions, the structure is as follows:

- Action of $\mathcal{Z}$:

REAL: Description of what happens in the real world.

IDEAL: Description of what happens in the ideal world.

IND: Argument why the worlds are indistinguishable.

Possible actions of $\mathcal{Z}$:

- (**register**, sid) to $\mathcal{F}_{\text{KRR}}$ in the name of corrupted $\mathcal{P}$.
- (**retrieve**, sid) to $\mathcal{F}_{\text{KRR}}$ in the name of $\mathcal{P}$.
- (**commit**, sid, m) to honest $\mathcal{C}$.
- (**open**, sid) to honest $\mathcal{C}$.
- (**sendOK**, $sid, \mathcal{C}, \mathcal{V}, msg$) to $\mathcal{V}$ in the name of honest or corrupted $\mathcal{C}$.

Preamble:

- Dummy adversary always forwards everything unchanged to $\mathcal{Z}$.
- Sim simulates the adversary in the ideal world. Everything going to $\mathcal{A}$ goes directly to Sim , where Sim has to simulate $\mathcal{A}$'s dummy behavior.
- Since we only consider static corruption, we treat the corruption cases separately.
- Remember that both the committer and verifier use the base commitment Π_{COM} as part of their protocol.
- Our simulator in Figure 3 makes use of a simulator Sim_{COM} which simulates Π_{COM} . This simulator is guaranteed to exist, because $\Pi_{\text{COM}} \geq_{\text{UC}} \mathcal{F}_{\text{COM}}$.
- The case “Committer whatever, Verifier malicious” is omitted. because it does not matter if $\mathcal{V}$ is corrupted or not: $\mathcal{V}$ does not send anything and the view is already simulated on the way via the public communication channels.
- All calls of $\mathcal{Z}$ to hybrid functionality $\mathcal{G}$ of Π_{COM} are handled by the internal Sim_{COM} . Both worlds are indistinguishable as Sim_{COM} is valid simulator for Π_{COM} .

Both parties are honest:

- (**register**, sid) to $\mathcal{F}_{\text{KRR}}$ in the name of corrupted $\mathcal{P}$:
Not applicable, because both parties are honest.

- (**retrieve**, sid) to $\mathcal{F}_{\text{KRR}}$ in the name of $\mathcal{P}$:

REAL: $\mathcal{F}_{\text{KRR}}$ is executed.

IDEAL: Sim internally runs $\mathcal{F}_{\text{KRR}}$ faithfully.

IND: Obviously indistinguishable because the same code is run in both worlds.

- (**commit**, sid , m) to honest $\mathcal{C}$:

REAL: $\mathcal{C}$ registers $sk_{\mathcal{C}}, pk_{\mathcal{C}}$ with $\mathcal{F}_{\text{KRR}}$, and generates σ_1 by signing sid and σ_2 by signing (sid, m) via USig under $sk_{\mathcal{C}}$. $\mathcal{C}$ samples randomness r . $\mathcal{C}$ then calls the commit interface of the base commitment Π_{COM} with $(\text{sid}, m; r)$ to get the commit information c . $\mathcal{C}$ stores the signature σ_2 and sends (**send**, sid , $\mathcal{C}$, $\mathcal{V}$, (σ_1, c)) to $\mathcal{A}$, who forwards it to $\mathcal{Z}$.

IDEAL: $\mathcal{C}$ forwards (**commit**, sid , m) to $\mathcal{F}_{\text{COM}}$. $\mathcal{F}_{\text{COM}}$ sends (**deliver committed**, sid) to Sim . Sim generates $(\overline{sk_{\mathcal{C}}}, \overline{pk_{\mathcal{C}}})$ according to the code of $\mathcal{F}_{\text{KRR}}$ and $\overline{\sigma_1} \leftarrow \text{USig.Sign}(\overline{sk_{\mathcal{C}}}, \text{sid})$. Sim internally runs Sim_{COM} with input (**deliver committed**, sid) in the name of the internal $\mathcal{F}_{\text{COM}}$ and gets the output (**send**, sid , $\mathcal{C}$, $\mathcal{V}$, $\overline{c}$) as output from Sim_{COM} to $\mathcal{Z}$. Sim stores $(\overline{sk_{\mathcal{C}}}, \overline{pk_{\mathcal{C}}}, \overline{\sigma_1}, \overline{c})$ and sends (**send**, sid , $\mathcal{C}$, $\mathcal{V}$, $(\overline{\sigma_1}, \overline{c})$) to $\mathcal{Z}$.

IND: It holds that $c \approx \overline{c}$ because Sim_{COM} properly simulates by assumption (hiding property). Additionally, $\sigma_1 \approx \overline{\sigma_1}$, because the keys of the signature and messages are indistinguishable between both worlds.

- (**open**, sid) to honest $\mathcal{C}$:

- REAL: $\mathcal{C}$ calls committer of Π_{COM} on its opening-phase with sid and gets the relevant opening information d . Then, $\mathcal{C}$ retrieves σ_2 and sends $(\text{send}, sid, \mathcal{C}, \mathcal{V}, (\sigma_2, d))$ to $\mathcal{A}$.
- IDEAL: $\mathcal{C}$ forwards (open, sid) to $\mathcal{F}_{\text{COM}}$. $\mathcal{F}_{\text{COM}}$ sends $(\text{deliver opened}, sid, m)$ to Sim . Sim instructs Sim_{COM} with $(\text{deliver opened}, sid, m)$ in the name of $\mathcal{F}_{\text{COM}}$ and receives $(\text{send}, sid, \mathcal{C}, \mathcal{V}, \bar{d})$ from Sim_{COM} as its output to $\mathcal{Z}$. Sim retrieves $\overline{sk}_{\mathcal{C}}$ and signs $\bar{\sigma}_2 \leftarrow \text{USig.Sign}(\overline{sk}_{\mathcal{C}}, (sid, m))$. Sim stores $\bar{\sigma}_2$ and m , and outputs $(\text{send}, sid, \mathcal{C}, \mathcal{V}, (\bar{\sigma}_2, \bar{d}))$ to $\mathcal{Z}$.
- IND: $d \approx \bar{d}$ again follows because Sim_{COM} properly simulates Π_{COM} and $\sigma_2 \approx \bar{\sigma}_2$ follows from the indistinguishability of the keys and messages.
- $(\text{sendOK}, sid, \mathcal{C}, \mathcal{V}, (\sigma_1, c))$ to $\mathcal{V}$ in the name of honest $\mathcal{C}$.
- REAL: $\mathcal{V}$ retrieves $pk_{\mathcal{C}}$ from $\mathcal{F}_{\text{KRK}}$ and stores it. $\mathcal{V}$ checks if $\text{USig.Vfy}(pk_{\mathcal{C}}, \sigma_1, sid)$ verifies. If it is the case, then $\mathcal{V}$ calls RcvCom of Π_{COM} with c . If this call returns $(\text{committed}, sid)$, then $\mathcal{V}$ does as well output $(\text{committed}, sid)$ to $\mathcal{Z}$.
- IDEAL: Sim runs the verifier code of Π on the instruction $(\text{sendOK}, sid, \mathcal{C}, \mathcal{V}, (\sigma_1, c))$. If the output is $(\text{committed}, sid)$, Sim sends $(\text{deliver committed}, sid)$ to $\mathcal{F}_{\text{COM}}$ in the name of $\mathcal{Z}$.
- IND: The verifier in the real world outputs $(\text{committed}, sid)$ if σ_1 is valid. In the ideal world the verifier outputs $(\text{committed}, sid)$ if σ_1 is valid and $\mathcal{Z}$ had sent (commit, sid, m) to $\mathcal{C}$. So the worlds are only distinguishable, if $\mathcal{Z}$ did not send (commit, sid, m) , but σ_1 is valid, in which case σ_1 is a signature forgery. Thus, by the EUF-CMA security of USig, both worlds are indistinguishable.
- $(\text{sendOK}, sid, \mathcal{C}, \mathcal{V}, (\sigma_2, d))$ to $\mathcal{V}$ in the name of honest $\mathcal{C}$.
- REAL: $\mathcal{V}$ retrieves $pk_{\mathcal{C}}$. $\mathcal{V}$ calls the verify-phase of base commitment Π_{COM} with d . If this call returns $(\text{opened}, sid, \bar{m})$, and if $\text{USig.Vfy}(pk_{\mathcal{C}}, \sigma_2, (sid, \bar{m})) = 1$, then $\mathcal{V}$ outputs $(\text{opened}, sid, \bar{m})$ to $\mathcal{Z}$.
- IDEAL: Sim runs the verifier code of Π on the instruction $(\text{sendOK}, sid, \mathcal{C}, \mathcal{V}, (\sigma_2, d))$. If the code returns $(\text{opened}, sid, \bar{m})$, Sim checks if $\bar{m}$ equals the message m that Sim previously received from $\mathcal{F}_{\text{COM}}$ as opening. If so, Sim sends $(\text{deliver opened}, sid)$ to $\mathcal{F}_{\text{COM}}$ in the name of $\mathcal{Z}$.
- IND: If σ_2 is not a valid signature for $(sid, \bar{m})$ then $\mathcal{V}$ outputs nothing in both worlds. If σ_2 is valid and $m = \bar{m}$, then $\mathcal{V}$ outputs the same message in both worlds. The real and ideal world are only distinguishable if σ_2 is valid and $m \neq \bar{m}$, in which case $(sid, \bar{m})$ is a fresh signature forgery that can be used to win the EUF-CMA game. Thus, by the EUF-CMA security of USig, both worlds are indistinguishable.

Committer malicious, Verifier whatever:

- $(\text{register}, sid)$ to $\mathcal{F}_{\text{KRK}}$ in the name of corrupted $\mathcal{P}$.
- REAL: $\mathcal{F}_{\text{KRK}}$ is executed.

- IDEAL: *Sim* generates fake $(\overline{sk_{\mathcal{P}}}, \overline{pk_{\mathcal{P}}})$ and stores it.
- IND: The fake keys are generated honestly via $\mathcal{F}_{\text{KRR}}$'s code, thereby it is obviously indistinguishable as the same code runs in both worlds.
- (**retrieve**, *sid*) to $\mathcal{F}_{\text{KRR}}$ in the name of $\mathcal{P}$.
- REAL: $\mathcal{F}_{\text{KRR}}$ is executed.
- IDEAL: *Sim* internally runs $\mathcal{F}_{\text{KRR}}$ faithfully.
- IND: Obviously indistinguishable because the same code is run in both worlds.
- (**commit**, *sid*, *m*) to honest $\mathcal{C}$.
Not applicable, because we are in the case where $\mathcal{C}$ is corrupted.
 - (**open**, *sid*) to honest $\mathcal{C}$.
Not applicable, because we are in the case where $\mathcal{C}$ is corrupted.
 - (**sendOK**, *sid*, $\mathcal{C}$, $\mathcal{V}$, (σ_1, c)) to $\mathcal{V}$ in the name of corrupted $\mathcal{C}$.
- REAL: $\mathcal{V}$ retrieves $pk_{\mathcal{C}}$ from $\mathcal{F}_{\text{KRR}}$ and stores it. If the signature σ_1 verifies, then $\mathcal{V}$ calls the commit-phase of the base commitment Π_{COM} with *c*. If this call outputs (**committed**, *sid*), then $\mathcal{V}$ outputs as well (**committed**, *sid*) to $\mathcal{Z}$.
- IDEAL: *Sim* retrieves $\overline{pk_{\mathcal{C}}}$ from storage. *Sim* verifies if $\text{USig.Vfy}(\overline{pk_{\mathcal{C}}}, \sigma_1, \text{sid}) = 1$. *Sim* instructs *Sim*_{COM} with (**sendOK**, *sid*, *c*) in the name of $\mathcal{F}_{\text{COM}}$ and receives (**commit**, *sid*, *m*) from *Sim*_{COM} as input of the malicious $\mathcal{C}$ to $\mathcal{F}_{\text{COM}}$. *Sim* stores *m* and sends (**commit**, *sid*, *m*) to $\mathcal{F}_{\text{COM}}$ in the name of corrupted $\mathcal{C}$. $\mathcal{F}_{\text{COM}}$ sends (**committed**, *sid*) to $\mathcal{V}$, which outputs it directly to $\mathcal{Z}$ as it is a dummy party in the ideal world.
- IND: Both in the real and ideal world $\mathcal{V}$ only outputs (**committed**, *sid*) if σ_1 is valid. And in that case the ideal $\mathcal{V}$ outputs the same message as the real $\mathcal{V}$, because *Sim*_{COM} properly simulates (in this case, because *Sim*_{COM} extracts the correct message).
- (**sendOK**, *sid*, $\mathcal{C}$, $\mathcal{V}$, (σ_2, d)) to $\mathcal{V}$ in the name of corrupted $\mathcal{C}$.
- REAL: $\mathcal{V}$ retrieves $pk_{\mathcal{C}}$. $\mathcal{V}$ calls the verify-phase of base commitment Π_{COM} with *d*. If this call outputs (**opened**, *sid*, *m*), and if $\text{USig.Vfy}(pk_{\mathcal{C}}, \sigma_2, (\text{sid}, m)) = 1$, then $\mathcal{V}$ outputs as well (**opened**, *sid*, *m*) to $\mathcal{Z}$.
- IDEAL: *Sim* retrieves *m* and $\overline{pk_{\mathcal{C}}}$ from storage. Then, *Sim* verifies if $\text{USig.Vfy}(\overline{pk_{\mathcal{C}}}, \sigma_2, (\text{sid}, m)) = 1$. *Sim* instructs *Sim*_{COM} with (**sendOK**, *sid*, *d*) in the name of $\mathcal{F}_{\text{COM}}$. If *Sim*_{COM} returns (**open**, *sid*) as message to $\mathcal{F}_{\text{COM}}$, and forwards it to $\mathcal{F}_{\text{COM}}$ in the name of corrupted $\mathcal{C}$. $\mathcal{F}_{\text{COM}}$ sends (**opened**, *sid*, *m*) to $\mathcal{V}$, which outputs it directly to $\mathcal{Z}$ as it is a dummy party in the IDEAL.
- IND: In both worlds $\mathcal{V}$ only outputs something, if σ_2 verifies. Then, in both worlds the output must be the same, because otherwise, *Sim*_{COM} did not extract the correct message in the commit phase. However, by the assumption that $\Pi_{\text{COM}} \geq_{\text{UC}} \mathcal{F}_{\text{COM}}$, there is a simulator *Sim*_{COM} that works correctly, thus, both worlds are indistinguishable.

This concludes the proof.

C Coin Toss without Authenticated Channel (Continued)

C.1 Routing of Messages with Internal Simulator

In the following, we give more detail on how *Sim* routes the messages between $Sim_{\Pi_{CT}}$, the simulated $\mathcal{P}_1$, $\mathcal{Z}$ and $\mathcal{F}_{CT}$ in the proof of Theorem 2. Whenever $Sim_{\Pi_{CT}}$ produces a message to inform $\mathcal{Z}$ about a message that was sent via $\mathcal{F}_{AUTH}$, *Sim* forwards that message to $\mathcal{Z}$ as if it were coming from an unauthenticated channel. Whenever $\mathcal{Z}$ instructs *Sim* to deliver a message to $\mathcal{P}_1$, *Sim* gives this message to its internal simulation of $\mathcal{P}_1$. When the simulated $\mathcal{P}_1$ sends a message to $\mathcal{P}_0$, *Sim* forwards this message to $\mathcal{Z}$. When $\mathcal{Z}$ instructs *Sim* to deliver a message to $\mathcal{P}_0$, *Sim* instructs $Sim_{\Pi_{CT}}$ to send this message to $\mathcal{P}_0$ in the name of the malicious $\mathcal{P}_1$. This works because *Sim* told $Sim_{\Pi_{CT}}$ that $\mathcal{P}_1$ is malicious. *Sim* also passes along messages between $\mathcal{F}_{CT}$ and $Sim_{\Pi_{CT}}$, except for the `(deliver, sid,  $\mathcal{P}$ )` instruction. There, *Sim* first simulates the exchange of the signatures and only sends `(deliver, sid,  $\mathcal{P}$ )` to $\mathcal{F}_{CT}$, if the counterparty's signature was valid.

C.2 Proof of Theorem 2

We show that no PPT environment $\mathcal{Z}$ can distinguish between the real world, where it is interacting with the parties and dummy adversary, from the ideal world, where it is interacting with the parties and the simulator. Figure 5 depicts the coin-tossing compiler protocol $\Pi := \Pi_{CT-comp}$. Figure 8 depicts the simulator $Sim := Sim_{\Pi}$. More formally, we show that

$$\exists Sim \quad \forall \mathcal{Z} : \quad EXEC_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa, r) \approx EXEC_{\mathcal{F}_{CT}, Sim, \mathcal{Z}}(\kappa, r),$$

where κ is the security parameter, r the randomness, and $\mathcal{A}$ the dummy adversary.

Note that messages typed as *msg* are messages handled in Π_{CT} and the last messages consisting of σ are messages of $\Pi_{CT-comp}$. We assume that *Sim* can distinguish between messages of Π_{CT} and the signatures σ . In the case that σ looks like a Π_{CT} message, one can slightly modify our protocol and prepend a 0 to the Π_{CT} messages and a 1 to the signatures, so that *Sim* can distinguish them.

For more clarity, we use an explicit formulation of $\mathcal{F}_{unAUTH}$ (Definition 2) as unauthenticated channel throughout this section. We prove the theorem by going through all actions that $\mathcal{Z}$ can take and arguing for each of them that $\mathcal{Z}$ cannot distinguish between real and ideal world. For each of $\mathcal{Z}$'s actions, the structure is as follows:

– Action of $\mathcal{Z}$:

REAL: Description of what happens in the real world.

IDEAL: Description of what happens in the ideal world.

IND: Argument why the worlds are indistinguishable.

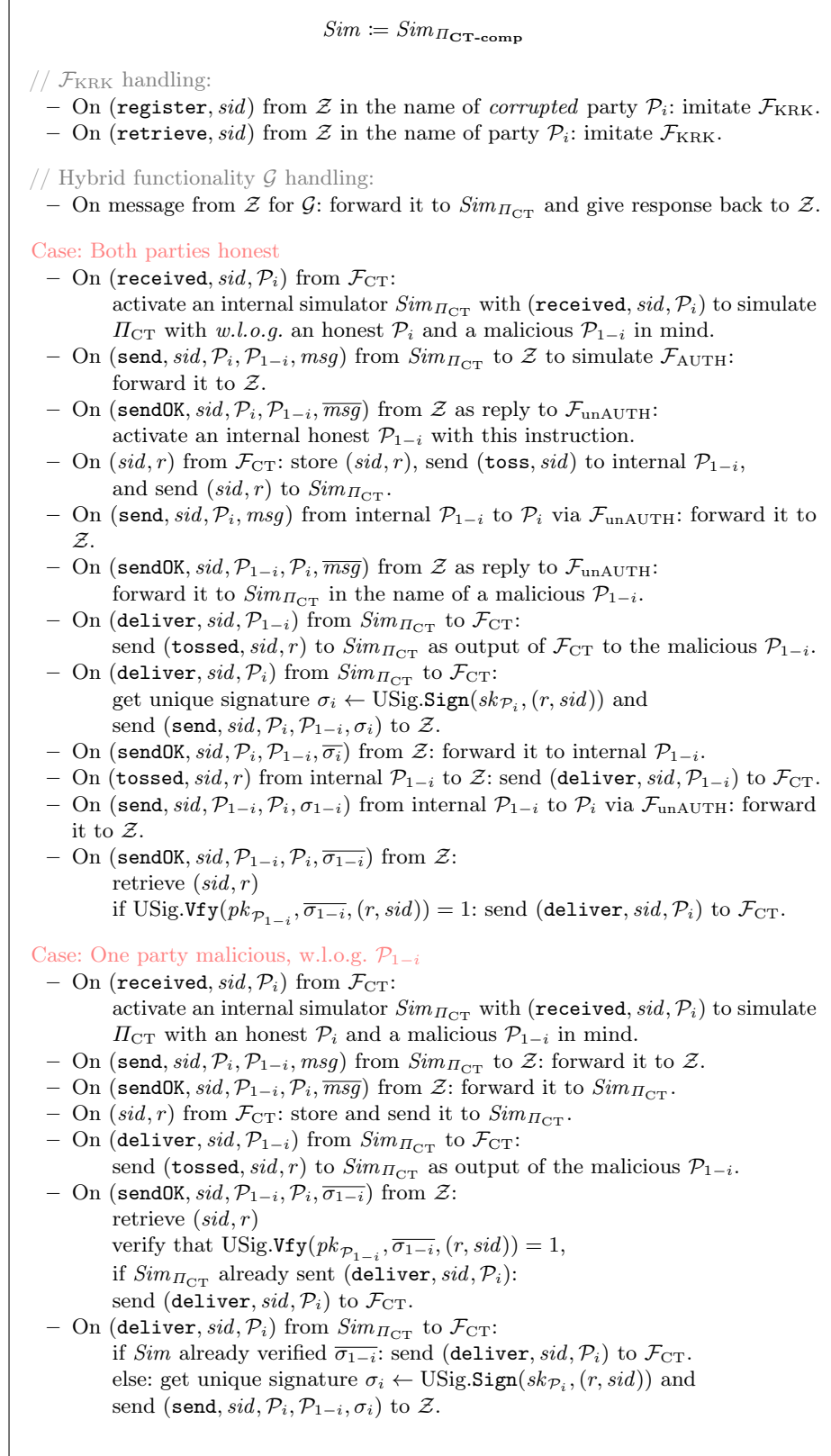


Fig. 8: Simulator for our coin-tossing protocol compiler from Figure 5.

Possible actions of $\mathcal{Z}$:

- (**register**, sid) to $\mathcal{F}_{\text{KRK}}$ in the name of corrupted $\mathcal{P}$.
- (**retrieve**, sid) to $\mathcal{F}_{\text{KRK}}$ in the name of $\mathcal{P}$.
- (**toss**, sid) to honest $\mathcal{P}$.
- (**sendOK**, $sid, \mathcal{P}_i, \mathcal{P}_{1-i}, msg$) as a message to $\mathcal{F}_{\text{unAUTH}}$.
- (**sendOK**, $sid, \mathcal{P}_i, \mathcal{P}_{1-i}, \sigma$) as a message to $\mathcal{F}_{\text{unAUTH}}$.
- (**sendOK**, $sid, \mathcal{P}_{1-i}, \mathcal{P}_i, msg$) as a message to $\mathcal{F}_{\text{unAUTH}}$.
- (**sendOK**, $sid, \mathcal{P}_{1-i}, \mathcal{P}_i, \sigma$) as a message to $\mathcal{F}_{\text{unAUTH}}$.

Preamble:

- Dummy adversary always forwards everything unchanged to $\mathcal{Z}$.
- Sim simulates the adversary in the ideal world. Everything going to $\mathcal{A}$ goes directly to Sim , where Sim has to simulate $\mathcal{A}$'s dummy behavior.
- Since we only consider static corruption, we treat the corruption cases separately.
- Remember that both parties use the base coin-tossing protocol Π_{CT} as part of their protocol.
- Our simulator in Figure 8 makes use of a simulator $Sim_{\Pi_{\text{CT}}}$ which simulates Π_{CT} . This simulator is guaranteed to exist, because $\Pi_{\text{CT}} \geq_{\text{UC}} \mathcal{F}_{\text{CT}}$.
- All calls of $\mathcal{Z}$ to hybrid functionality $\mathcal{G}$ of Π_{CT} are handled by the internal $Sim_{\Pi_{\text{CT}}}$. Both worlds are indistinguishable as $Sim_{\Pi_{\text{CT}}}$ is a valid simulator for Π_{CT} .

One party honest, one malicious: w.l.o.g. $\mathcal{P}_{1-i}$

- (**register**, sid) to $\mathcal{F}_{\text{KRK}}$ in the name of corrupted $\mathcal{P}_{1-i}$.
 REAL: $\mathcal{F}_{\text{KRK}}$ is executed.
 IDEAL: Sim generates fake $(\overline{sk_{\mathcal{P}_{1-i}}}, \overline{pk_{\mathcal{P}_{1-i}}})$ and stores it.
 IND: The fake keys are generated honestly via $\mathcal{F}_{\text{KRK}}$'s code, thereby it is obviously indistinguishable as the same code runs in both worlds.
- (**retrieve**, sid) to $\mathcal{F}_{\text{KRK}}$ in the name of $\mathcal{P}$.
 REAL: $\mathcal{F}_{\text{KRK}}$ is executed.
 IDEAL: Sim internally runs $\mathcal{F}_{\text{KRK}}$'s code faithfully.
 IND: Obviously indistinguishable as the same code runs in both worlds.
- (**toss**, sid) to honest $\mathcal{P}_i$.
 REAL: $\mathcal{P}_i$ forwards (**toss**, sid) as input for honest $\mathcal{P}_i$ in the base coin-tossing protocol Π_{CT} . As Π_{CT} is in the $\mathcal{F}_{\text{AUTH}}$ -hybrid model, it will send an input to $\mathcal{F}_{\text{AUTH}}$, which $\mathcal{P}_i$ takes and sends via $\mathcal{F}_{\text{unAUTH}}$ (which means that $\mathcal{Z}$ gets the message).
 IDEAL: $\mathcal{F}_{\text{CT}}$ sends (**received**, $sid, \mathcal{P}_i$) to Sim , who forwards it to $Sim_{\Pi_{\text{CT}}}$. As $Sim_{\Pi_{\text{CT}}}$ simulates Π_{CT} , it outputs a simulated $\mathcal{F}_{\text{AUTH}}$ -message to its environment being Sim , which forwards it to $\mathcal{Z}$ as output of $\mathcal{F}_{\text{unAUTH}}$.

IND: No messages is handled in between and in both worlds $\mathcal{Z}$ receives $(\text{send}, \text{sid}, \mathcal{P}_i, \mathcal{P}_{1-i}, \text{msg})$ as output of $\mathcal{F}_{\text{unAUTH}}$, with msg being the first message from $\mathcal{P}_i$ to $\mathcal{P}_{1-i}$. Therefore, both worlds are indistinguishable.

- $(\text{sendOK}, \text{sid}, \mathcal{P}_i, \mathcal{P}_{1-i}, \text{msg})$ as message to $\mathcal{F}_{\text{unAUTH}}$. This action is meaningless, as $\mathcal{P}_{1-i}$ is malicious and $\mathcal{Z}$ would only send a message to itself.
- $(\text{sendOK}, \text{sid}, \mathcal{P}_{1-i}, \mathcal{P}_i, \text{msg})$ as message to $\mathcal{F}_{\text{unAUTH}}$, where msg is a message in the base coin-tossing protocol Π_{CT} .

REAL: $\mathcal{P}_i$ forwards the message as a message of $\mathcal{F}_{\text{AUTH}}$ to Π_{CT} . If Π_{CT} produces another message from $\mathcal{P}_i$ to $\mathcal{P}_{1-i}$ as output for $\mathcal{F}_{\text{AUTH}}$, it is given again to $\mathcal{Z}$ as output of $\mathcal{F}_{\text{unAUTH}}$. Else if Π_{CT} outputs $(\text{tossed}, \text{sid}, r)$ as output for $\mathcal{Z}$, $\mathcal{P}_i$ generates $\sigma_i \leftarrow \text{USig.Sign}(sk_{\mathcal{P}_i}, (r, \text{sid}))$ and sends $(\text{send}, \text{sid}, \mathcal{P}_i, \mathcal{P}_{1-i}, \sigma_i)$ to $\mathcal{Z}$.

IDEAL: Sim forwards the message as a message to $\mathcal{F}_{\text{AUTH}}$ (from the malicious $\mathcal{P}_{1-i}$) to $\text{Sim}_{\Pi_{\text{CT}}}$. If $\text{Sim}_{\Pi_{\text{CT}}}$ produces another message from $\mathcal{P}_i$ to $\mathcal{P}_{1-i}$ as output for $\mathcal{F}_{\text{AUTH}}$, it is given again to $\mathcal{Z}$ as output of $\mathcal{F}_{\text{unAUTH}}$. Else if $\text{Sim}_{\Pi_{\text{CT}}}$ sends $(\text{deliver}, \mathcal{P}_i)$ as input for $\mathcal{F}_{\text{CT}}$, Sim generates $\sigma_i \leftarrow \text{USig.Sign}(sk_{\mathcal{P}_i}, (r, \text{sid}))$ and sends $(\text{send}, \text{sid}, \mathcal{P}_i, \mathcal{P}_{1-i}, \sigma_i)$ to $\mathcal{Z}$.

IND: In both worlds $\mathcal{Z}$ receives either $\mathcal{P}_i$'s next Π_{CT} -message, or a valid signature on r from $\mathcal{P}_i$. Therefore, both worlds are indistinguishable.

- $(\text{sendOK}, \text{sid}, \mathcal{P}_i, \mathcal{P}_{1-i}, \sigma_i)$ as a message to $\mathcal{F}_{\text{unAUTH}}$. This action is meaningless, because $\mathcal{Z}$ can ping-pong any message to $\mathcal{P}_{1-i}$ in itself as it is malicious.
- $(\text{sendOK}, \text{sid}, \mathcal{P}_{1-i}, \mathcal{P}_i, \sigma_{1-i})$ as message to $\mathcal{F}_{\text{unAUTH}}$.

REAL: $\mathcal{P}_i$ verifies that $\text{USig.Vfy}(pk_{\mathcal{P}_{1-i}}, \overline{\sigma_{1-i}}, (r, \text{sid})) = 1$, and if Π_{CT} already output $(\text{tossed}, \text{sid}, r)$: $\mathcal{P}_i$ outputs $(\text{tossed}, \text{sid}, r)$ to $\mathcal{Z}$.

IDEAL: Sim verifies that $\text{USig.Vfy}(pk_{\mathcal{P}_{1-i}}, \overline{\sigma_{1-i}}, (r, \text{sid})) = 1$, and if $\text{Sim}_{\Pi_{\text{CT}}}$ already sent $(\text{deliver}, \text{sid}, \mathcal{P}_i)$: Sim sends $(\text{deliver}, \text{sid}, \mathcal{P}_i)$ to $\mathcal{F}_{\text{CT}}$.

IND: On $(\text{deliver}, \text{sid}, \mathcal{P}_i)$, if $\mathcal{F}_{\text{CT}}$ already sampled r , it sends $(\text{tossed}, \text{sid}, r)$ to $\mathcal{P}_i$, which forwards it directly to $\mathcal{Z}$. Therefore, both worlds are indistinguishable.

Both parties honest

- $(\text{register}, \text{sid})$ to $\mathcal{F}_{\text{KRK}}$ in the name of corrupted $\mathcal{P}_{1-i}$.
Not applicable, because both parties are honest.

- $(\text{retrieve}, \text{sid})$ to $\mathcal{F}_{\text{KRK}}$ in the name of $\mathcal{P}$.

REAL: $\mathcal{F}_{\text{KRK}}$ is executed.

IDEAL: Sim internally runs $\mathcal{F}_{\text{KRK}}$'s code faithfully.

IND: Obviously indistinguishable as the same code runs in both worlds.

- (**toss**, sid) to honest $\mathcal{P}_i$.
 - REAL: $\mathcal{P}_i$ forwards (**toss**, sid) as input for honest $\mathcal{P}_i$ in base coin-tossing protocol Π_{CT} . As Π_{CT} is in the $\mathcal{F}_{AUTH}$ -hybrid model, it will send an input to $\mathcal{F}_{AUTH}$, which $\mathcal{P}_i$ takes and sends via $\mathcal{F}_{unAUTH}$ (which means that $\mathcal{Z}$ gets the message).
 - IDEAL: $\mathcal{F}_{CT}$ sends (**received**, $sid, \mathcal{P}_i$) to Sim , who forwards it to $Sim_{\Pi_{CT}}$. As $Sim_{\Pi_{CT}}$ simulates Π_{CT} , it outputs a simulated $\mathcal{F}_{AUTH}$ -message to its environment being Sim , which forwards it to $\mathcal{Z}$ as output of $\mathcal{F}_{unAUTH}$.
 - IND: No messages is handled in between and in both worlds $\mathcal{Z}$ receives (**send**, $sid, \mathcal{P}_i, \mathcal{P}_{1-i}, msg$) as output of $\mathcal{F}_{unAUTH}$, with msg being the first message from $\mathcal{P}_i$ to $\mathcal{P}_{1-i}$. Therefore, both worlds are indistinguishable.
-
- (**sendOK**, $sid, \mathcal{P}_i, \mathcal{P}_{1-i}, msg$) as message to $\mathcal{F}_{unAUTH}$, where msg is a message in the base coin-tossing protocol Π_{CT} .
 - REAL: $\mathcal{P}_{1-i}$ forwards the message as message of $\mathcal{F}_{AUTH}$ to Π_{CT} . If Π_{CT} produces another message from $\mathcal{P}_{1-i}$ to $\mathcal{P}_i$ as output for $\mathcal{F}_{AUTH}$, it is given again to $\mathcal{Z}$ as output of $\mathcal{F}_{unAUTH}$. Else if Π_{CT} outputs (**tossed**, sid, r) as output for $\mathcal{Z}$, $\mathcal{P}_{1-i}$ generates $\sigma_{1-i} \leftarrow \text{USig.Sign}(sk_{\mathcal{P}_{1-i}}, (r, sid))$ and sends (**send**, $sid, \mathcal{P}_{1-i}, \mathcal{P}_i, \sigma_{1-i}$) to $\mathcal{F}_{unAUTH}$ and, thereby, to $\mathcal{Z}$.
 - IDEAL: Sim gives the action to its internal honest $\mathcal{P}_{1-i}$. If $\mathcal{P}_{1-i}$ produces another message to $\mathcal{P}_i$, which is either a message msg for Π_{CT} via $\mathcal{F}_{AUTH}$ or a signature σ_{1-i} . In both cases, it is given again to $\mathcal{Z}$ as message from $\mathcal{F}_{unAUTH}$.
 - IND: In both worlds $\mathcal{Z}$ receives either another message of the format (**send**, $sid, \mathcal{P}_{1-i}, \mathcal{P}_i, msg$) or (**send**, $sid, \mathcal{P}_{1-i}, \mathcal{P}_i, \sigma_{1-i}$), where the σ_{1-i} is generated faithfully in both worlds. Therefore, both worlds are indistinguishable.
-
- (**sendOK**, $sid, \mathcal{P}_{1-i}, \mathcal{P}_i, msg$) as a message to $\mathcal{F}_{unAUTH}$ where msg is a message in the base coin-tossing protocol Π_{CT} .
 - REAL: $\mathcal{P}_i$ forwards the message to Π_{CT} (as message from $\mathcal{F}_{AUTH}$). If Π_{CT} produces another message from $\mathcal{P}_i$ to $\mathcal{P}_{1-i}$ as output for $\mathcal{F}_{AUTH}$, it is given again to $\mathcal{Z}$ as output of $\mathcal{F}_{unAUTH}$. Else if Π_{CT} outputs (**tossed**, sid, r) as output for $\mathcal{Z}$, $\mathcal{P}_i$ generates $\sigma_i \leftarrow \text{USig.Sign}(sk_{\mathcal{P}_i}, (r, sid))$ and sends (**send**, $sid, \mathcal{P}_i, \mathcal{P}_{1-i}, \sigma_i$) to $\mathcal{Z}$.
 - IDEAL: Sim forwards the message as a message of $\mathcal{F}_{AUTH}$ to $Sim_{\Pi_{CT}}$ in the name of a malicious $\mathcal{P}_{1-i}$. If $Sim_{\Pi_{CT}}$ produces another message from $\mathcal{P}_i$ to $\mathcal{P}_{1-i}$ as output for $\mathcal{F}_{AUTH}$, it is given again to $\mathcal{Z}$ as output of $\mathcal{F}_{unAUTH}$. Else if $Sim_{\Pi_{CT}}$ outputs (**deliver**, $sid, \mathcal{P}_i$) as input for $\mathcal{F}_{CT}$, Sim generates $\sigma_i \leftarrow \text{USig.Sign}(sk_{\mathcal{P}_i}, (r, sid))$ and sends (**send**, $sid, \mathcal{P}_i, \mathcal{P}_{1-i}, \sigma_i$) to $\mathcal{Z}$.
 - IND: In both worlds $\mathcal{Z}$ receives (**send**, $sid, \mathcal{P}_i, \mathcal{P}_{1-i}, \sigma_i$) as output of $\mathcal{F}_{unAUTH}$ and with a freshly generated signature for $\mathcal{P}_i$. Therefore, both worlds are indistinguishable.
-
- (**sendOK**, $sid, \mathcal{P}_i, \mathcal{P}_{1-i}, \sigma_i$) as message to $\mathcal{F}_{unAUTH}$.
 - REAL: $\mathcal{P}_{1-i}$ verifies that $\text{USig.Vfy}(pk_{\mathcal{P}_i}, \bar{\sigma}_i, (r, sid)) = 1$, and if Π_{CT} already output (**tossed**, sid, r): $\mathcal{P}_{1-i}$ outputs (**tossed**, sid, r) to $\mathcal{Z}$.

- IDEAL: *Sim* forwards the message to its internal honest party $\mathcal{P}_{1-i}$. If $\mathcal{P}_{1-i}$ outputs $(\text{tossed}, \text{sid}, r)$, *Sim* sends $(\text{deliver}, \text{sid}, \mathcal{P}_{1-i})$ to $\mathcal{F}_{\text{CT}}$.
- IND: On $(\text{deliver}, \text{sid}, \mathcal{P}_i)$, if $\mathcal{F}_{\text{CT}}$ already sampled r , it sends $(\text{tossed}, \text{sid}, r)$ to $\mathcal{P}_i$, which forwards it directly to $\mathcal{Z}$ as it a dummy party in the ideal world. Therefore, both worlds are indistinguishable.
- $(\text{sendOK}, \text{sid}, \mathcal{P}_{1-i}, \mathcal{P}_i, \sigma_{1-i})$ as message to $\mathcal{F}_{\text{unAUTH}}$ to $\mathcal{P}_i$.
- REAL: $\mathcal{P}_i$ verifies that $\text{USig.Vfy}(pk_{\mathcal{P}_{1-i}}, \overline{\sigma_{1-i}}, (r, \text{sid})) = 1$, and if Π_{CT} already output $(\text{tossed}, \text{sid}, r)$: $\mathcal{P}_i$ outputs $(\text{tossed}, \text{sid}, r)$ to $\mathcal{Z}$.
- IDEAL: *Sim* verifies that $\text{USig.Vfy}(pk_{\mathcal{P}_{1-i}}, \overline{\sigma_{1-i}}, (r, \text{sid})) = 1$, and if *Sim* $_{\Pi_{\text{CT}}}$ already sent $(\text{deliver}, \text{sid}, \mathcal{P}_i)$: *Sim* sends $(\text{deliver}, \text{sid}, \mathcal{P}_i)$ to $\mathcal{F}_{\text{CT}}$.
- IND: On $(\text{deliver}, \text{sid}, \mathcal{P}_i)$, if $\mathcal{F}_{\text{CT}}$ already sampled r , it sends $(\text{tossed}, \text{sid}, r)$ to $\mathcal{P}_i$, which forwards it directly to $\mathcal{Z}$ as it a dummy party in the ideal world. Therefore, both worlds are indistinguishable.

D Revisiting Subversion UC (Continued)

D.1 Some Supporting Lemmas

We state a few lemmas which are useful when working with our model of subversion-resilient UC (i.e., the integration of our subversion notion in ABMO). The lemmas will be applied in Section 5.

Lemma 3. *For protocols that do not allow subversion corruptions, UC-emulation and srUC-emulation are equivalent:*

$$\Pi \geq_{\text{UC}} \Phi \iff \Pi \geq_{\text{srUC}} \Phi.$$

Proof (sketch). As both Π and Φ do not allow subversion corruptions, it does not matter that srUC-emulations restricts these corruptions to specious ones.

Lemma 4. *Adding the dummy reveal-secret interface does not affect srUC-emulation:*

$$\Pi \geq_{\text{srUC}} \Phi \iff \Pi^r \geq_{\text{srUC}} \Phi^r.$$

This also holds if Φ is an ideal functionality.

Proof (sketch). For “ $\Leftarrow$ ” the same simulator that worked for $\Pi^r \geq_{\text{UC}} \Phi^r$ also works for $\Pi \geq_{\text{UC}} \Phi$. For “ $\Rightarrow$ ” we need to slightly adapt the simulator *Sim* of $\Pi \geq_{\text{UC}} \Phi$ so that it also works for $\Pi^r \geq_{\text{UC}} \Phi^r$. Namely when *Sim* gets notified that the dummy **reveal-secret** interface has been called for a party $\mathcal{P}$, then it stops to simulate messages for $\mathcal{P}$.

Lemma 5. *Standard UC-emulation is equivalent to srUC-emulation with dummy reveal-secret interfaces:*

$$\Pi \geq_{\text{UC}} \Phi \iff \Pi^r \geq_{\text{srUC}} \Phi^r.$$

Proof (sketch). This lemma follows from combining Lemma 4 and Lemma 3.

Lemma 6 (Transitivity of srUC-emulation). *Let Π, Φ, P be protocols (which may allow subversion corruptions or have the dummy reveal-secret interface). Then, the following implication holds:*

$$\Pi \geq_{\text{srUC}} \Phi \text{ and } \Phi \geq_{\text{srUC}} P \implies \Pi \geq_{\text{srUC}} P.$$

Proof (sketch). As the simulator is restricted in the same way as the environment is (i.e., only subversion corruptions with specious code are allowed), transitivity holds the same way as in the original UC framework [14, Sec. 4.5].

E Subversion- and Impersonation-Resilient Protocols (Continued)

E.1 Transparency and Anti-Signaling of RF'

Lemma 7. *If RF is a strongly transparent firewall (see Definition 7) for Π_{COM} , then RF' is a strongly transparent firewall for $\Pi_{\text{COM-comp}}$ (as defined in Figure 6).*

Proof (sketch). Since RF is strongly transparent and RF' leaves the signatures unmodified, it is indistinguishable for an environment whether it is interacting with the committer of $\Pi_{\text{COM-comp}}$, or with the committer of Π_{COM} together with RF' . Thus, RF' provides *outer transparency*. Additionally, the committer in $\Pi_{\text{COM-comp}}$ runs the same code, regardless of whether a firewall is present. Thus, RF' provides *strong transparency*.

Lemma 8. *If RF is an anti-signaling firewall (see Definition 8) for Π_{COM} , then RF' is an anti-signaling firewall for $\Pi_{\text{COM-comp}}$ (as defined in Figure 6).*

Proof (sketch). For $\Pi_{\text{COM-comp}}$, we need to show that the following two executions are indistinguishable for any specious environment: (i) an execution with an incorruptible committer and an incorruptible firewall, and (ii) an execution with a subvertible committer and an incorruptible firewall. Because RF is anti-signaling for Π_{COM} , the commitment and opening information c and d are indistinguishable in both cases. Furthermore, the signatures σ_1, σ_2 are unique signatures. If the subverted code would alter them (relative to the honest code), then due to the uniqueness property, they would not verify anymore. This is detectable by a test and, thus, a specious code cannot alter the signatures. Therefore, RF' is anti-signaling for specious corruptions.

E.2 Proof of Theorem 3

Proof (sketch). Because we only consider static corruptions, we can prove the statement for all different cases of corruptions separately. Note that in all the corruption cases below of the committer and its firewall, it does not matter whether the verifier is honest or malicious. Additionally, a subverted verifier

must behave honestly, because otherwise it would not be specious according to Definition 1: If the subverted verifier deviates from honest behavior (either by sending a message to the network, or by outputting an incorrect value), then there is an efficient test that can detect this behavior. Even if there are only a few messages from the committer that would trigger such a dishonest behavior, the test can detect this, as it is non-uniform and can get one such message as auxiliary input. Thus, what remains is to go through all corruption cases of the committer $\mathcal{C}$ and its firewall RF' :

- Honest $\mathcal{C}$ and honest RF' (treated as honest $\mathcal{C}$ in $\Pi_{COM-comp}^r$)
- Honest $\mathcal{C}$ and malicious RF' (treated as honest $\mathcal{C}$ in $\Pi_{COM-comp}^r$)
- Subverted $\mathcal{C}$ and honest RF' (treated as honest $\mathcal{C}$ in $\Pi_{COM-comp}^r$)
- Malicious $\mathcal{C}$ and honest RF' (treated as malicious $\mathcal{C}$ in $\Pi_{COM-comp}^r$)
- Malicious $\mathcal{C}$ and malicious RF' (treated as malicious $\mathcal{C}$ in $\Pi_{COM-comp}^r$)
- Honest $\mathcal{C}$ and semi-honest RF' (treated as honest $\mathcal{C}$ in $\Pi_{COM-comp}^r$)

Honest $\mathcal{C}$ and honest RF' in Π maps to honest $\mathcal{C}$ in $\Pi_{COM-comp}^r$: Here, the simulator Sim does not have to do anything. As RF' provides (outer) transparency, an execution of the protocol with honest committer and honest firewall is indistinguishable from an execution with an honest committer, which is exactly what we want to show in this case.

Honest $\mathcal{C}$ and malicious RF' in Π maps to honest $\mathcal{C}$ in $\Pi_{COM-comp}^r$: The high level argument in this case is simple: The malicious firewall has the same power as the network adversary, so it cannot do any extra harm. In a bit more detail, when the simulator (in $\Pi_{COM-comp}^r$) gets a message from $\mathcal{C}$ (via the unauthenticated channel), it hands it to $\mathcal{Z}$ as a message from $\mathcal{C}$ to RF' . Furthermore, whenever $\mathcal{Z}$ instructs the malicious firewall to send a certain message m , Sim sends m via the unauthenticated channel (with the same sender and recipient as specified by $\mathcal{Z}$).

Subverted $\mathcal{C}$ and honest RF' in Π maps to honest $\mathcal{C}$ in $\Pi_{COM-comp}^r$: This case is also rather simple due to the preparatory work: The execution with a subverted $\mathcal{C}$ and an honest RF' is indistinguishable to the execution with an honest $\mathcal{C}$ and an honest RF' as shown in Lemma 8. Furthermore, due to the transparency of RF' (Lemma 7), this is indistinguishable from the execution with an honest $\mathcal{C}$. Thus, in this case the simulator, does not need to do anything.

Malicious $\mathcal{C}$ and honest RF' in Π maps to malicious $\mathcal{C}$ in $\Pi_{COM-comp}^r$: When the simulator gets from $\mathcal{Z}$ the instruction to send a message m in the name of the malicious committer, Sim internally runs RF' on m and sends the resulting message to the network.

Malicious $\mathcal{C}$ and malicious RF' in Π maps to malicious $\mathcal{C}$ in $\Pi_{COM-comp}^r$: When the simulator gets from $\mathcal{Z}$ the instruction to send a message m in the name of the malicious committer, Sim hands back m to $\mathcal{Z}$ as the message that the malicious RF' received. When the simulator gets from $\mathcal{Z}$ the instruction to send a message m in the name of the malicious firewall, Sim sends m to the network.

Honest $\mathcal{C}$ and semi-honest RF' in Π maps to honest $\mathcal{C}$ in $\Pi_{COM-comp}^r$: When the simulator receives a message m from the honest $\mathcal{C}$, it hands m to $\mathcal{Z}$ as a message from $\mathcal{C}$ to RF' . Upon instruction from $\mathcal{Z}$ to continue the execution, Sim runs RF' on m and sends the resulting message to the network. This simulation works because the committer in Π^s and the committer in $\Pi_{COM-comp}^r$ run the same code, as the RF' is strongly transparent.

E.3 Proof of Theorem 4

Proof (sketch). The proof is similar to the proof of Theorem 3 and we cover the same combinations of corruptions of $\mathcal{P}_i$ and its firewall. We only show the corruption cases for $\mathcal{P}_i$, since the roles of the parties are symmetric and the simulation works regardless of the corruption status of the other party.

Honest $\mathcal{P}_i$ and honest RF'_i in Π maps to honest $\mathcal{P}_i$ in $\Pi_{CT-comp}^r$: Analogous to the proof of Theorem 3, the simulator does not do anything and indistinguishability holds because of (outer) transparency.

Honest $\mathcal{P}_i$ and malicious RF'_i in Π maps to honest $\mathcal{P}_i$ in $\Pi_{CT-comp}^r$: This case is also analogous to the proof of Theorem 3, since the firewall essentially takes the role of the network adversary.

Subverted $\mathcal{P}_i$ and honest RF'_i in Π maps to honest $\mathcal{P}_i$ in $\Pi_{CT-comp}^r$: Since RF'_i is strongly transparent and anti-signaling, this corruption case is indistinguishable from the execution with honest $\mathcal{P}_i$.

Malicious $\mathcal{P}_i$ and honest RF'_i in Π maps to malicious $\mathcal{P}_i$ in $\Pi_{CT-comp}^r$: Again, this case is analogous to the proof of Theorem 3.

Malicious $\mathcal{P}_i$ and malicious RF'_i in Π maps to malicious $\mathcal{P}_i$ in $\Pi_{CT-comp}^r$: Again, this case is analogous to the proof of Theorem 3.

Honest $\mathcal{P}_i$ and semi-honest RF'_i in Π maps to honest $\mathcal{P}_i$ in $\Pi_{CT-comp}^r$: Again, this case is analogous to the proof of Theorem 3.