# Universally Composable Subversion-Resilient Cryptography

Suvradip Chakraborty*[1], Bernardo Magri†[2], Jesper Buus Nielsen[3], and Daniele Venturi‡[4]

[1]ETH Zurich
[2]University of Manchester
[3]Aarhus University
[4]Sapienza University of Rome

February 25, 2022

**Abstract**

Subversion attacks undermine security of cryptographic protocols by replacing a legitimate honest party's implementation with one that leaks information in an undetectable manner. An important limitation of all currently known techniques for designing cryptographic protocols with security against subversion attacks is that they do not automatically guarantee security in the realistic setting where a protocol session may run concurrently with other protocols.

We remedy this situation by providing a foundation of *reverse firewalls* (Mironov and Stephens-Davidowitz, EUROCRYPT'15) in the *universal composability* (UC) framework (Canetti, FOCS'01 and J. ACM'20). More in details, our contributions are threefold:

- We generalize the UC framework to the setting where each party consists of a core (which has secret inputs and is in charge of generating protocol messages) and a firewall (which has no secrets and sanitizes the outgoing/incoming communication from/to the core). Both the core and the firewall can be subject to different flavors of corruption, modeling different kinds of subversion attacks. For instance, we capture the setting where a subverted core looks like the honest core to any efficient test, yet it may leak secret information via covert channels (which we call *specious subversion*).
- We show how to sanitize UC commitments and UC coin tossing against specious subversion, under the DDH assumption.
- We show how to sanitize the classical GMW compiler (Goldreich, Micali and Wigderson, STOC 1987) for turning MPC with security in the presence of semi-honest adversaries into MPC with security in the presence of malicious adversaries. This yields a completeness theorem for maliciously secure MPC in the presence of specious subversion.

Additionally, all our sanitized protocols are *transparent*, in the sense that communicating with a sanitized core looks indistinguishable from communicating with an honest core. Thanks to the composition theorem, our methodology allows, for the first time, to design subversion-resilient protocols by sanitizing different sub-components in a modular way.

*Work done while at IST Austria. The author is supported in part by ERC grant 724307.

†Work done while at Aarhus University.

‡Supported by the grant SPECTRA from Sapienza University of Rome.

# Contents

# 1 Introduction

Cryptographic schemes are typically analyzed under the assumption that the machines run by honest parties are fully trusted. Unfortunately, in real life, there are a number of situations in which this assumption turns out to be false. In this work, we are concerned with one of these situations, where the adversary is allowed to subvert the implementation of honest parties in a stealthy way. By stealthy, we mean that the outputs produced by a subverted machine still look like honestly computed outputs, yet, the adversary can use such outputs to completely break security. Prominent examples include backdoored implementations [DGG+15, DPSW16, FJM18] and algorithm-substitution (or *kleptographic*) attacks [YY96, YY97, BPR14, BJK15, BL17]. The standardization of the pseudorandom number generator Dual_EC_DRBG, as exposed by Snowden, is a real-world instantiation of the former, while Trojan horses, as in the case of the Chinese hack chip attack, are real-world instantiations of the latter.

## 1.1 Subversion-Resilient Cryptography

Motivated by these situations, starting from the late 90s, cryptographers put considerable effort into building cryptographic primitives and protocols that retain some form of security in the presence of *subversion attacks*. We review the state of the art in more details in Section 1.4.

Yet, after nearly 30 years of research, all currently known techniques to obtain subversion resilience share the limitation of only implying *standalone* security, *i.e.* they only guarantee security of a protocol in isolation, but all bets are off when such a protocol is used in a larger context in the presence of subversion attacks. This shortcoming makes the design of subversion-resilient cryptographic protocols somewhat cumbersome and highly non-modular. For instance, Ateniese, Magri, and Venturi [AMV15] show how to build subversion-resilient signatures, which in turn were used by Dodis, Mironov and Stephens-Davidowitz [DMS16] to obtain subversion-resilient key agreement protocols, and by Chakraborty, Dziembowski and Nielsen [CDN20] to obtain subversion-resilient broadcast; however, the security analysis in both [DMS16] and [CDN20] reproves security of the construction in [AMV15] from scratch. These examples bring the fundamental question:

*Can we obtain subversion resistance in a* composable *security framework?*

A positive answer to the above question would dramatically simplify the design of subversion-resilient protocols, in that one could try to first obtain security under subversion attacks for simpler primitives, and then compose such primitives in an arbitrary way to obtain protocols for more complex tasks, in a modular way.

## 1.2 Our Contributions

In this work, we give a positive answer to the above question using so-called *cryptographic reverse firewalls*, as introduced by Mironov and Stephens-Davidowitz [MS15]. Intuitively, a reverse firewall is an external party that sits between an honest party and the network, and whose task is to sanitize the incoming/outgoing communication of the party it is attached to, in order to annhilate subliminal channels generated via subversion attacks. The main challenge is to obtain sanitation while maintaining the correctness of the underlying protocol, and in a setting where other parties may be completely under control of the subverter itself.

While previous work showed how to build reverse firewalls for different cryptographic protocols in *standalone* security frameworks (see Section 1.4), we provide a foundation of reverse firewalls in the framework of universal composability (UC) of Canetti [Can01, Can00]. More in details, our contributions are threefold:

- We generalize the UC framework to the setting where each party consists of a core (which has secret inputs and is in charge of generating protocol messages) and a firewall (which has no secrets and sanitizes the outgoing/incoming communication from/to the core). Both the core and the firewall can be subject to different flavors of corruption, modeling different kinds of subversion attacks. For instance, we capture the setting where a subverted core looks like the honest core to any efficient test, yet it may leak secret information via covert channels (which we call *specious subversion*).

- We show how to sanitize UC commitments and UC coin tossing against specious subversion, under the decisional Diffie-Hellman (DDH) assumption in the common reference string (CRS) model. Our sanitized commitment protocol is non-interactive, and requires $2\lambda$ group elements in order to commit to a $\lambda$-bit string; the CRS is made of 3 group elements.

- We show how to sanitize the classical compiler by Goldreich, Micali and Wigderson (GMW) [GMW87] for turning multiparty computation (MPC) with security against semi-honest adversaries into MPC with security against malicious adversaries. This yields a completeness theorem for maliciously secure MPC in the presence of specious subversion.

Additionally, all our sanitized protocols are *transparent*, in the sense that communicating with a sanitized core looks indistinguishable from communicating with an honest core. Thanks to the composition theorem, our methodology allows, for the first time, to design subversion-resilient protocols by sanitizing different sub-components in a modular way.

## 1.3 Technical Overview

Below, we provide an overview of the techniques we use in order to achieve our results, starting with the notion of subversion-resilient UC security, and then explaining the main ideas behind our reverse firewalls constructions.

### 1.3.1 Subversion-resilient UC Security

At a high level we model each logical party $\mathsf{P}_i$ of a protocol $\Pi$ as consisting of two distinct parties of the UC framework, one called the core $\mathsf{C}_i$ and one called the firewall $\mathsf{F}_i$. These parties can be independently corrupted. For instance, the core can be subverted and the firewall honest, or the core could be honest and the firewall corrupted. The ideal functionalities $\mathcal{F}$ implemented by such a protocol will also recognize two UC parties per virtual party and can let their behavior depend on the corruption pattern. For instance, $\mathcal{F}$ could specify that if $\mathsf{C}_i$ is subverted and $\mathsf{F}_i$ honest, then it behaves as if $\mathsf{P}_i$ is honest on $\mathcal{F}$. Or it could say that if $\mathsf{C}_i$ is honest and $\mathsf{F}_i$ corrupt, then it behaves as if $\mathsf{P}_i$ is honest but might abort on $\mathcal{F}$. This is a reasonable choice as a corrupt firewall can always cut $\mathsf{C}_i$ off from the network and force an abort. We then simply ask that $\Pi$ UC-realizes $\mathcal{F}$. By asking that $\Pi$ UC-realizes $\mathcal{F}$ we exactly capture that if the core is subverted and the firewall is honest, this has the same effect as $\mathsf{P}_i$ being honest. See Table 1 for all possible corruption combinations for $\mathsf{C}_i$ and $\mathsf{F}_i$ at a glance, and how they translate into corruptions for $\mathsf{P}_i$ in an ideal execution with functionality $\mathcal{F}$.

Unfortunately, it turns out that for certain functionalities it is just impossible to achieve security in the presence of *arbitrary* subversion attacks. For instance, a subverted prover in a zero-knowledge proof could simply output an honestly computed proof or the all-zero string depending on the first bit of the witness. Since the firewall would not know a valid witness, these kind of subversion attacks cannot be sanitized. For this reason, following previous work [MS15, DMS16, GMV20, CDN20], we focus on classes of subversion attacks for which a subverted core looks like an honest core to any efficient test, yet it may signal private information to the subverter via subliminal channels. We call such corruptions *specious*. We note that testing reasonably models a scenario in which the core has been built by an untrusted manufacturer who wants to stay covert, and where the user tests it against a given specification before using it in the wild.

By defining subversion resilience in a black-box way, via the standard notion of UC implementation, we also get composition almost for free via the UC composition theorem. One complication arises to facilitate modular composition of protocols. When doing a modular construction of a subversion-resilient protocol, both the core and the firewall will be built by modules. For instance, the core could be built from a core for a commitment scheme and the core for an outer protocol using the commitment scheme. Each of these cores will come with their own firewall: one sanitizes the core of the commitment scheme; the other sanitizes the core of the outer protocol. The overall firewall is composed of these two firewalls. It turns out that it is convenient that these two firewalls can coordinate, as it might be that some of the commitments sent need to have the message randomized, while others might only have their randomness refreshed. The latter can be facilitated by giving the firewall of the commitment scheme a sanitation interface where it can be instructed by the outer firewall to do the right sanitation. Note that the protocol implementing the commitment ideal functionality now additionally needs to implement this sanitation interface.

We refer the reader to Section 2 for a formal description of our model. Note that another natural model would have been to have $\mathsf{P}_i$ split into three parts (or tiers), $\mathsf{C}_i$, $\mathsf{U}_i$, and $\mathsf{F}_i$, where: (i) $\mathsf{U}_i$ is a user program which gets inputs and sends messages on the network; (ii) $\mathsf{C}_i$ is a core holding cryptographic keys and implementations of, *e.g.*, signing and encryption algorithms; and (iii) $\mathsf{F}_i$ is a firewall used by $\mathsf{U}_i$ to sanitize messages to and from $\mathsf{C}_i$ in order to avoid covert channels. The above better models a setting where we are only worried that some part of the computer might be subverted. The generalisation to this case is straightforward given the methodology we present for the case with no user program $\mathsf{U}_i$. Since we only look at subversions which are indistinguishable from honest implementations, having the "unsubvertable" $\mathsf{U}_i$ appears to give no extra power. We therefore opted for the simpler model for clarity. Further discussion on the three-tier model can be found in Appendix A.

**Strong sanitation.** The main challenge when analyzing subversion security of a protocol in our framework is that, besides maliciously corrupting a subset of the parties, the adversary can, *e.g.*, further speciously corrupt the honest parties. To overcome this challenge, we introduce a simple property of reverse firewalls which we refer to as *strong sanitation*. Intuitively, this property says that no environment, capable of doing specious corruptions of an honest core in the real world, can distinguish an execution of the protocol with one where an honest core is replaced with a so-called *incorruptible* core (that simply behaves honestly in case of specious corruption). The latter, of course, requires that the firewall of the honest core is honest.

We then prove a general lemma saying that, whenever a firewall has strong sanitation, it is enough to prove security in our model without dealing with specious corruptions of honest parties. This lemma significantly simplifies the security analysis of protocols in our model.

### 1.3.2 Commitments

In Section 3, we show how to obtain subversion-resilient UC commitments. First, we specify a sanitizable string commitment functionality $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$. This functionality is basically identical to the standard functionality for UC commitments [CF01], except that the firewall is allowed to sanitize the value $s$ that the core commits to, using a blinding factor $r$; the effect of this sanitation is that, when the core opens the commitment, the ideal functionality reveals $\hat{s} = s \oplus r$.

Note that this is the sanitation allowed by the sanitation interface. An implementation will further have to sanitise the randomness of outgoing commitments to avoid covert channels.

Second, we construct a protocol $\widehat{\Pi}_{\mathsf{sCOM}}$ that UC realizes $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$ in the presence of subversion attacks. Our construction borrows ideas from a recent work by Canetti, Sarkar and Wang [CSW20], who showed how to construct efficient non-interactive UC commitments with adaptive security. The protocol, which is in the CRS model and relies on the standard DDH assumption, roughly works as follows. The CRS is a tuple of the form $(g, h, T_1, T_2)$, such that $T_1 = g^x$ and $T_2 = h^{x'}$ for $x \neq x'$ (*i.e.*, a non-DH tuple). In order to commit to a single bit $b$, the core of the committer encodes $b$ as a value $u \in \{-1, 1\}$ and outputs $B = g^\alpha \cdot T_1^u$ and $H = h^\alpha \cdot T_2^u$, where $\alpha$ is the randomness. The firewall sanitizes a pair $(B, H)$ by outputting $\widehat{B} = B^{-1} \cdot g^\beta$ and $\widehat{H} = H^{-1} \cdot h^\beta$, where $\beta$ is chosen randomly; note that, upon receiving an opening $(b, \alpha)$ from the core, the firewall can adjust it by returning $(1 - b, -\alpha + \beta)$. Alternatively, the firewall can choose to leave the bit $b$ unchanged and only refresh the randomness of the commitment; this is achieved by letting $\widehat{B} = B \cdot g^\beta$ and $\widehat{H} = H \cdot h^\beta$; in this case, the opening is adjusted to $(b, \alpha + \beta)$. In the security proof, we distinguish two cases:

- In case the committer is maliciously corrupt, the simulator sets the CRS as in the real world but additionally knows the discrete log $t$ of $h$ to the base $g$. Such a trapdoor allows the simulator to extract the bit $b$ corresponding to the malicious committer by checking whether $H/T_2 = (B/T_1)^t$ (in which case $b = 1$) or $H \cdot T_2 = (B \cdot T_1)^t$ (in which case $b = 0$). If none of the conditions hold, no opening exists.

- In case the committer is honest, the simulator sets the CRS as a DH-tuple. Namely, now $T_1 = g^x$ and $T_2 = h^x$ for some $x$ known to the simulator. The latter allows the simulator to fake the commitment as $B = g^\alpha$ and $H = h^\alpha$, and later adjust the opening to any given $u \in \{-1, 1\}$ (and thus $b \in \{0, 1\}$) by letting $\alpha' = \alpha - u \cdot x$.

The above ideas essentially allow to build a simulator for the case of two parties, where one is maliciously corrupt and the other one has an honest core and a semi-honest firewall. These ideas can be generalized to $n$ parties (where up to $n - 1$ parties are maliciously corrupt, while the remaining party has an honest core and a semi-honest firewall) using an independent CRS for each pair of parties. Finally, we show that the firewall in our protocol is strongly sanitizing and thus all possible corruption cases reduce to the previous case. In particular, strong sanitation holds true because a specious core must produce a pair $(B, H)$ of the form $B = g^\alpha \cdot T_1^{\tilde{u}}$ and $H = h^\alpha \cdot T_2^{\tilde{u}}$ for some $\tilde{u} \in \{-1, 1\}$ (and thus $\tilde{b} \in \{0, 1\}$), as otherwise a tester could distinguish it from an honest core by asking it to open the commitment; given such a well-formed commitment, the firewall perfectly refreshes its randomness (and eventually blinds the message).

As we show in Section 3, the above construction can be extended to the case where the input to the commitment is a $\lambda$-bit string by committing to each bit individually; the same CRS can be reused across all of the commitments.

### 1.3.3 Coin Tossing

Next, in Section 4, we show a simple protocol that UC realizes the standard coin tossing functionality $\mathcal{F}_{\mathsf{TOSS}}$ in the presence of subversion attacks. Recall that the ideal functionality $\mathcal{F}_{\mathsf{TOSS}}$ samples a uniformly random string $s \in \{0, 1\}^\lambda$ and sends it to the adversary, which can then decide which honest party gets $s$ (*i.e.*, the coin toss output).

Our construction is a slight variant of the classical coin tossing protocol by Blum [Blu81]; the protocol is in the $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$-hybrid model, and roughly works as follows. The core of each party commits to a random string $s_i \in \{0, 1\}^\lambda$ through the ideal functionality $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$. Then, the firewall of the coin toss instructs the firewall of the commitment to blind $s_i$ using a random blinding factor $r_i \in \{0,1\}^\lambda$ which is revealed to the core. At this point, each (willing) party opens the commitment, which translates into $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$ revealing $\hat{s}_j = s_j \oplus r_j$, and each party finally outputs $s = s_i \oplus r_i \oplus \bigoplus_{j \neq i} \hat{s}_j$.

In the security proof, the simulator can fake the string $s_i$ of an honest party so that it matches the output of the coin tossing $s$ (received from $\mathcal{F}_{\mathsf{TOSS}}$), the strings $s_j$ received from the adversary (on behalf of a malicious core), and the blinding factor $r_i$ received from the adversary (on behalf of a semi-honest firewall). This essentially allows to build a simulator for the case where up to $n-1$ parties are maliciously corrupt, while the remaining party has an honest core and a semi-honest firewall. Finally, we show that the firewall in our protocol is strongly sanitizing and thus all possible corruption cases reduce to the previous case. Strong sanitation here holds because any string $s_i$ chosen by a specious core is mapped to a uniformly random string $\hat{s}_i$ via the sanitation interface of the functionality $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$.

### 1.3.4 Completeness Theorem

Finally, in Section 5, we show how to sanitize the GMW compiler, which yields a completeness theorem for UC subversion-resilient MPC. Recall that in the classical GMW compiler one starts with an MPC protocol $\Pi$ tolerating $t < n$ semi-honest corruptions and transforms it into an MPC protocol tolerating $t$ malicious corruptions as follows. First, the players run an augmented coin-tossing protocol, where each party receives a uniformly distributed string (to be used as its random tape) and the other parties receive a commitment to that string. Second, each party commits to its own input and proves in zero knowledge that every step of the protocol $\Pi$ is executed correctly and consistently with the random tape and input each party is committed to.

As observed by Canetti, Lindell, Ostrovsky and Sahai [CLOS02], the above compilation strategy cannot immediately be translated in the UC setting, as the receiver of a UC commitment obtains no information about the value that was committed to. Hence, the parties cannot prove in zero knowledge statements relative to their input/randomness commitment. This issue is resolved by introducing a commit-and-prove ideal functionality, which essentially allows each party to commit to a witness and later prove arbitrary NP statements relative to the committed witness.

In order to sanitize the GMW compiler in the presence of subversion attacks, we follow a similar approach. Namely, we first introduce a sanitazable commit-and-prove functionality $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. This functionality is very similar in spirit to the standard commit-and-prove functionality, except that the firewall can decide to blind the witness that the core commits to. In Appendix B, we show how to realize the sanitizable commit-and-prove functionality in the CRS model from the DDH assumption, using re-randomizable non-interactive zero-knowledge arguments for all of NP [CKLM12]. In fact, there we exhibit a much more general construction that can be instantiated from any so-called *malleable mixed commitment*, a new notion that we introduce and that serves as a suitable abstraction of our DDH-based construction from Section 3.

In the actual protocol, we use both the coin tossing functionality $\mathcal{F}_{\mathsf{TOSS}}$ and the sanitizable commit-and-prove functionality $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$ to determine the random tape of each party as follows. Each core commits to a random string $s_i$ via $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$; the corresponding firewall blinds $s_i$ with a random $r_i$ that is revealed to the core. Thus, the players use $\mathcal{F}_{\mathsf{TOSS}}$ to generate public randomness $s_i^*$ that can be used to derive the random tape of party $\mathsf{P}_i$ as $s_i^* \oplus (s_i \oplus r_i)$. Moreover each core commits to its own input $x_i$, which however is not blinded by the firewall. The above allows each party, during the protocol execution, to prove via $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$ that each message has been computed correctly and consistently with the committed input and randomness derived from the public random string $s_i^*$ received from $\mathcal{F}_{\mathsf{TOSS}}$.

The security analysis follows closely the one in [CLOS02], except that in our case we show that any adversary corrupting up to $t$ parties maliciously, and the firewall of the remaining honest parties semi-honestly, can be reduced to a semi-honest adversary attacking $\Pi$. Since we additionally show that our firewall is strongly sanitizing, which essentially comes from the ideal sanitation interface offered by $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$, all possible corruption cases reduce to the previous case.

### 1.4 Related Work

Below, we review state-of-the-art tools for obtaining subversion resilience and compare them to our approach, also explaining why they fall short of obtaining any composable security guarantees.

**Reverse firewalls.** In their original paper, Mironov and Stephens-Davidowitz [DMS16] show how to construct reverse firewalls for oblivious transfer (OT) and two-party computation with semi-honest security. Follow-up research showed how to construct reverse firewalls for a plethora of cryptographic primitives and protocols including: secure message transmission and key agreement [DMS16, CMY$^+$16, BBF$^+$20], interactive proof systems [GMV20], and maliciously secure MPC for both the case of static [CDN20] and adaptive [CGPS21] corruptions.

Most of these constructions are highly non-modular, as the security of each firewall is proven in isolation and thus it does not carry over when the firewall is composed with other firewalls in a larger protocol. For instance, Mironov and Stephens-Davidowitz [DMS16] combine reverse firewalls for re-randomizable garbled circuits with reverse firewalls for OT in order to obtain a reverse firewall for two-party computation with semi-honest security; however, the security analysis of their two-party protocol re-proves security of the OT reverse firewall from scratch.

Exceptions are the works by Dodis, Mironov and Stephens-Davidowitz [DMS16], Chakraborty, Dziembowski and Nielsen [CDN20], and Chakraborty, Ganesh, Pancholi and Sarkar [CGPS21]. In particular, these works do construct reverse firewalls for certain primitives and then combine those firewalls in larger protocols without re-proving their security from scratch. However, this is achieved each time by proving a composition theorem which is *protocol specific*, and thus only works for a particular way in which reverse firewalls are combined. This is in sharp contrast with our generalization of the UC composition theorem, which instead allows a protocol designer to construct subversion-resilient protocols for a given functionality and then use that protocol as a sub-component of *any* larger protocol, even when run concurrently with other protocols, and while still guaranteeing security in the presence of subversion attacks.

**Watchdogs.** The line of research on *cliptography* [RTYZ16, RTYZ17, RTYZ18, CRT$^+$19, AFMV19, CHY20, BCJ21] shows how to clip the power of subversion attacks without the need for reverse firewalls, and assuming only that a watchdog algorithm can perform a black-box test to decide whether a (possibly subverted) implementation is compliant to its specification. All of these works only consider game-based security, and thus provide no composition guarantee.

A watchdog can be *offline* (meaning that testing only happens before a scheme is deployed), *online* (meaning that testing is executed in parallel to the deployment of the scheme), or *omniscient* (meaning that testing additionally depends on the implementation's secret state). Unfortunately, offline watchdogs alone are not powerful enough to detect pretty natural classes of subversion attacks such as input-triggered attacks [DFP15, AMV15]. In contrast, reverse firewalls allow to annihilate input-triggered attacks via offline testing *and* sanitation.

Note that offline watchdogs are similar in spirit to our testing algorithm in the definition of specious corruption; one difference is that in the watchdog model one additionally assumes that an algorithm is decomposed into several functional components (*e.g.*, one component for key generation, one for sampling randomness, and etc.), whereas we consider a party of an MPC protocol as consisting of a single core. Online watchdogs and omniscient watchdogs, instead, are both incomparable to reverse firewalls as the former needs to test a running implementation, and the latter needs access to secret inputs, but none of them sanitizes the protocol transcript.

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

**Further related work.** Additional work related to subversion security includes research on parameters subversion [BFS16, ABLZ17, Fuc18, ALSZ20], collusion-free protocols [LMs05, AsV08], subliminal channels [Sim84, Sim86], and divertible protocols [OO90, BDI$^+$99]. We refer the reader to [MS15] for a comprehensive comparison of these works with reverse firewalls.

## 2 A UC Model of Reverse Firewalls

In this section we propose a foundation of reverse firewalls in the UC model [Can01]. We use the UC framework for concreteness as it is the *de facto* standard. However, we keep the description high level and do not depend on very particular details of the framework. Similar formalizations could be given in other frameworks defining security via comparison to ideal functionalities, as long as these ideal functionalities are corruption aware: they know which parties are corrupted and their behavior can depend on it.

### 2.1 Quick and Dirty Recap of UC

A protocol $\Pi$ consists of code for each of the parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$. The parties can in turn make calls to ideal functionalities $\mathcal{G}$. More precisely, the code of the program is a single machine. As part of its input, it gets a party identifier $\mathsf{pid}$ which tells the code which party it should be running the code for. This allows more flexibility for dynamic sets of parties. Below, we will only consider programs with a fixed number of parties. We are therefore tacitly identifying $n$ parties identifiers $\mathsf{pid}_1, \ldots, \mathsf{pid}_n$ with the $n$ parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, *i.e.*, $\mathsf{P}_i = \mathsf{pid}_i$. We prefer the notation $\mathsf{P}_i$ for purely idiomatic reasons.

A party $\mathsf{P}_i$ can call an ideal functionality. To do so it will specify which $\mathcal{G}$ to call (technically it writes down the code of $\mathcal{G}$ and a session identifier $\mathsf{sid}$ distinguishing different calls), along with an input $x$. Then, $(\mathsf{sid}, \mathsf{pid}, x)$ is given to $\mathcal{G}$. If $\mathcal{G}$ does not exists, then it is created from its code.

There is an adversary $\mathcal{A}$ which attacks the protocol. It can corrupt parties via special corruption commands. How parties react to these corruptions is flexible; the parties can in principle be programmed to react in any efficient way. As an example, in response to input `active-corrupt`, we might say that the party in the future will output all its inputs to the adversary, and that it will let the adversary specify what messages the party should send. The adversary can also control ideal functionalities, if the ideal functionalities expose an interface for that. It might for instance be allowed to influence at what time messages are delivered on an ideal functionality of point-to-point message transmission.

There is also an environment $\mathcal{E}$ which gives inputs to the parties and sees their outputs. The environment can talk freely to the adversary. A real world execution $\textsc{Exec}_{\Pi,\mathcal{A},\mathcal{E}}$ is driven by the environment which can activate parties or ideal functionalities. The parties and ideal functionalities can also activate each other. The details of activation are not essential here, and can be found in [Can01].

The protocol $\Pi$ is meant to implement an ideal functionality $\mathcal{F}$. This is formulated by considering a run of $\mathcal{F}$ with dummy parties which just forward messages between $\mathcal{E}$ and $\mathcal{F}$. In addition, there is an adversary $\mathcal{S}$, called the simulator, which can interact with $\mathcal{F}$ on the adversarial interface, and which can interact freely with $\mathcal{E}$ as an adversary can. The simulation is the process $\textsc{Exec}_{\mathcal{F},\mathcal{S},\mathcal{E}}$, where we do not specify the dummy protocol but use $\mathcal{F}$ for the dummy protocol composed with $\mathcal{F}$. We say that $\Pi$ UC-realizes $\mathcal{F}$ if there exists an efficient simulator which makes the simulation look like the real world execution to any efficient environment:

$$\exists \mathcal{S} \forall \mathcal{E} : \textsc{Exec}_{\Pi,\mathcal{A},\mathcal{E}} \approx \textsc{Exec}_{\mathcal{F},\mathcal{S},\mathcal{E}},$$

where $\mathcal{A}$ is the dummy adversary (that simply acts as a proxy for the environment), and where the quantifications are over poly-time interactive Turing machines.

Consider a protocol $\Pi$ that realizes an ideal functionality $\mathcal{F}$ in a setting where parties can communicate as usual, and additionally make calls to an unbounded number of copies of some other ideal functionality $\mathcal{G}$. (This model is called the $\mathcal{G}$-hybrid model.) Furthermore, let $\Gamma$ be a protocol that UC-realizes $\mathcal{G}$ as sketched above, and let $\Pi^{\mathcal{G}\to\Gamma}$ be the composed protocol that is identical to $\Pi$, with the exception that each interaction with the ideal functionality $\mathcal{G}$ is replaced with a call to (or an activation of) an appropriate instance of the protocol $\Gamma$. Similarly, any output produced by the protocol $\Gamma$ is treated as a value provided by the functionality $\mathcal{G}$. The composition theorem states that in such a case, $\Pi$ and $\Pi^{\mathcal{G}\to\Gamma}$ have essentially the same input/output behavior. Namely, $\Gamma$ behaves just like the ideal functionality $\mathcal{G}$ even when composed with an arbitrary protocol $\Pi$. A special case of this theorem states that if $\Pi$ UC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, then $\Pi^{\mathcal{G}\to\Gamma}$ UC-realizes $\mathcal{F}$.

## 2.2 Modeling Reverse Firewalls

To model reverse firewalls, we will model each party $\mathsf{P}_i$ as two separate parties in the UC model: the core $\mathsf{C}_i$ and the firewall $\mathsf{F}_i$. To be able to get composability for our framework via UC composition, we model them as separate parties each with their own party identifier $(\mathsf{pid}, \mathtt{F})$ and $(\mathsf{pid}, \mathtt{C})$. We use $\mathsf{pid}$ to denote the two of them together. Below we write, for simplicity, $\mathsf{P}_i$ to denote the full party, $\mathsf{C}_i$ to denote the core, and $\mathsf{F}_i$ to denote the firewall. Being two separate parties, the core and the firewall cannot talk directly. It will be up to the ideal functionality $\mathcal{G}$ used for communication to pass communication with the core through the corresponding firewall before acting on the communication. It might be that when $\mathcal{G}$ gets a message from $\mathsf{C}_i$ it will output this message to $\mathsf{F}_i$ and allow $\mathsf{F}_i$ to change the message, possibly under some restrictions. We say that $\mathsf{F}_i$ sanitizes the communication, and we call the interface connecting $\mathsf{F}_i$ for $\mathcal{G}$ the sanitation interface of $\mathcal{G}$. We call such an ideal functionality a "sanitizable" ideal functionality.

Consider a party $(\mathsf{C}_i, \mathsf{F}_i)$ with core $\mathsf{C}_i$ and firewall $\mathsf{F}_i$ connected to a sanitizing ideal functionality $\mathcal{G}$. The idea is that the firewall gets to sanitize all communication of the core $\mathsf{C}_i$. The UC model seemingly allows a loophole, as the core could make a call to some other ideal functionality $\mathcal{H}$ instead of talking to $\mathcal{G}$. As we discuss later, this behavior is ruled out if $\mathsf{C}_i$ is specious, so we will not explicitly disallow it. If our model is later extended to allow stronger (non-specious) types of subversion, then one would probably have to explicitly forbid $\mathsf{C}_i$ to use this loophole.

When using a sanitizable ideal functionality, it is convenient to be able to distinguish the interface of the ideal functionality from the parties using the interface. We call the interface of $\mathcal{G}$ to which the core of $\mathsf{P}_i$ is connected the input-output interface, $\mathsf{IO}$. We call the party connected to it $\mathsf{C}_i$. We call the interface of $\mathcal{G}$ to which the firewall of $\mathsf{P}_i$ is connected the sanitation interface, $\mathsf{S}$. We call the party connected to it $\mathsf{F}_i$. This is illustrated in Fig. 1.

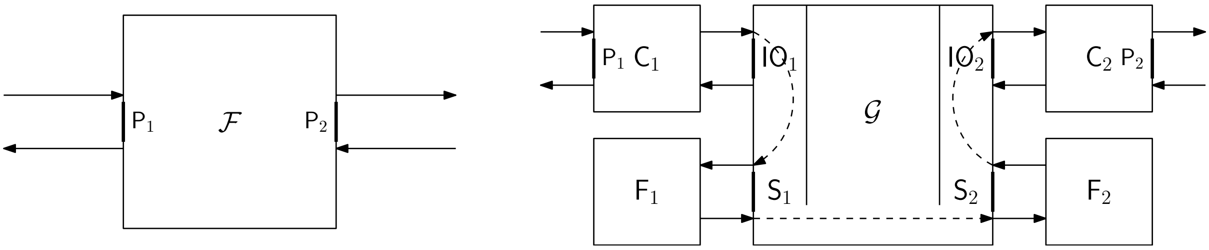


Figure 1: Implementing a normal functionality $\mathcal{F}$ using a sanitizable hybrid functionality $\mathcal{G}$ and a sanitizing protocol $\Pi = (\mathsf{C}, \mathsf{F})$. Cores and firewalls talk to sanitizable functionalities directly. Cores can additionally talk to the environment to exchange inputs and outputs. Firewalls only talk to ideal functionalities. We think of ideal functionalities as sanitizing the communication with the core via the firewall. This is illustrated in the figure by information from the core going to the firewall, and information to the core coming via the firewall. There is no formal requirement to what extent this happens; it is up to the ideal functionality to decide what type of sanitation is possible, if any.

### 2.3 Specious Corruptions

A major motivation for studying subversion resilience is to construct firewalls which ensure that security is preserved even if the core is subverted. In this section, we describe and discuss how we model subversion in the UC framework.

In a nutshell, we let the adversary replace the code of the core. Clearly, if the core is arbitrarily corrupted, it is impossible to guarantee any security. We therefore have to put restrictions on the code used to subvert the core. One can consider different types of subversions. In this work, we will consider a particularly "benign" subversion, where the subverted core looks indistinguishable from the honest core to any efficient test. This is a particularly strong version of what has been called "functionality preservation" in other works [MS15, DMS16, GMV20, CDN20]. As there are slightly diverting uses of this term we will coin a new one to avoid confusion.

The central idea behind our notion is that we consider corruptions where a core $\mathsf{C}_i$ has been replaced by another implementation $\widetilde{\mathsf{C}}_i$ which cannot be distinguished from $\mathsf{C}_i$ by black-box access to $\widetilde{\mathsf{C}}_i$ or $\mathsf{C}_i$. We use the term *specious* for such corruptions, as they superficially appear to be honest, but might not be.

More in details, we define specious corruptions via testing. Imagine a test $\mathsf{T}$ which is given non-rewinding black-box access to either $\mathsf{C}_i$ or $\widetilde{\mathsf{C}}_i$, and that tries to guess which one it interacted with. We say that a subversion is specious if it survives all efficient tests. This is a very strong notion. One way to motivate this notion could be that $\widetilde{\mathsf{C}}_i$ might be built by an untrusted entity, but the buyer of $\widetilde{\mathsf{C}}_i$ can test it up against a specification. If the untrusted entity wants to be sure to remain covert, it would have to do a subversion that survives all tests. We assume that the test does not have access to the random choices made by $\widetilde{\mathsf{C}}_i$. This makes the model applicable also to the case where $\widetilde{\mathsf{C}}_i$ is a blackbox or uses an internal physical process to make random choices. We will allow the entity doing the subversion to have some auxiliary information about the subversion and its use of randomness. This will, for instance, allow the subversion to communicate with the subverter in a way that cannot be detected by any test (*e.g.*, using a secret message acting as a trigger).

For a machine $\mathsf{T}$ and an interactive machine $\widetilde{\mathsf{C}}$, we use $\mathsf{T}^{\widetilde{\mathsf{C}}}$ to denote that $\mathsf{T}$ has non-rewinding black-box access to $\widetilde{\mathsf{C}}$. If during the run of $\mathsf{T}^{\widetilde{\mathsf{C}}}$ the machine $\widetilde{\mathsf{C}}$ requests a random bit, then a uniformly random bit is sampled and given to $\widetilde{\mathsf{C}}$. Such randomness is not shown to $\mathsf{T}$. We define the following game for an efficiently sampleable distribution $\mathsf{D}$ and a test $\mathsf{T}$.

- Sample $(\widetilde{\mathsf{C}}, a) \leftarrow \mathsf{D}$, where $a$ is an auxiliary string.
- Sample a uniformly random bit $b \in \{0,1\}$:
  - If $b = 0$, then run $\mathsf{T}^{\widetilde{\mathsf{C}}}$ to get a guess $g \in \{0,1\}$.
  - If $b = 1$, then run $\mathsf{T}^{\mathsf{C}}$ to get a guess $g \in \{0,1\}$.
- Output $c = b \oplus g$.

Let $\textsc{Test}_{\mathsf{D},\mathsf{T}}$ denote the probability that $c = 0$, *i.e.*, the probability that the guess at $b$ is correct.

**Definition 1** (Specious subversion)**.** We say that $\mathsf{D}$ is computationally specious if for all PPT tests $\mathsf{T}$ it holds that $\textsc{Test}_{\mathsf{D},\mathsf{T}} - 1/2$ is negligible.

We return to the discussion of the loophole for specious cores of creating other ideal functionalities $\mathcal{H}$ that are not sanitizing. Note that if a core creates an ideal functionality that it is not supposed to contact, then this can be seen by testing. Therefore, such a core is not considered specious. Hence, the notion of specious closes the loophole.

The notion of specious is strong, as it requires that no test $\mathsf{T}$ can detect the subversion. At first glance it might even look too strong, as it essentially implies that the subversion is correct. However, as we show next, a specious subversion can still signal to the outside in an undetectable manner. To formalize this notion, we define the following game for an efficiently sampleable distribution $\mathsf{D}$, an adversary $\mathsf{A}$ and a decoder $\mathsf{Z}$.

- Sample $(\widetilde{\mathsf{C}}, a) \leftarrow \mathsf{D}$, where $a$ is an auxiliary string.
- Sample a uniformly random bit $b \in \{0,1\}$:
  - If $b = 0$, then run $\mathsf{A}^{\widetilde{\mathsf{C}}}$ to get a signal $s \in \{0,1\}^*$.
  - If $b = 1$, then run $\mathsf{A}^{\mathsf{C}}$ to get a signal $s \in \{0,1\}^*$.
- Run $\mathsf{Z}(a, s)$ to get a guess $g \in \{0,1\}$.
- Output $c = b \oplus g$.

Let $\textsc{Signal}_{\mathsf{D},\mathsf{A},\mathsf{Z}}$ denote the probability that $c = 0$, *i.e.*, the probability that the guess at $b$ is correct.

**Definition 2** (Signaling)**.** We say that $\mathsf{D}$ is computationally signalling if there exists a PPT adversary $\mathsf{A}$ and a PPT decoder $\mathsf{Z}$ such that $\textsc{Signal}_{\mathsf{D},\mathsf{A},\mathsf{Z}} - 1/2$ is non-negligible.

**Lemma 1.** *There exist a machine* $\mathsf{C}$*, and an efficiently sampleable distribution* $\mathsf{D}$*, such that* $\mathsf{D}$ *is both computationally specious and signaling.*

*Proof sketch.* Consider a machine $\mathsf{C}$ that when queried outputs a fresh uniformly random $y \in \{0,1\}^\lambda$. Let $\Phi = \{\phi_\kappa : \{0,1\}^\lambda \to \{0,1\}^\lambda\}_{\kappa \in \{0,1\}^\lambda}$ be a family of pseudorandom permutations. Consider the subversion $\widetilde{\mathsf{C}}$ of $\mathsf{C}$ that hardcodes a key $\kappa \in \{0,1\}^\lambda$ and: (i) when initialised samples a uniformly random counter $x \in \{0,1\}^\lambda$; (ii) when queried, it returns $\phi_\kappa(x)$ and increments $x$. Moreover, let $\mathsf{D}$ be the distribution that picks $\kappa \in \{0,1\}^\lambda$ at random and outputs $(\widetilde{\mathsf{C}}, a = \kappa)$.

Note that the distribution $\mathsf{D}$ is specious, as the key $\kappa$ is sampled at random *after* $\mathsf{T}$ has been quantified. In particular, the outputs of $\phi_\kappa$ are indistinguishable from random to $\mathsf{T}$. The distribution $\mathsf{D}$ is also clearly signaling, as it can be seen by taking the adversary $\mathsf{A}$ that queries its target oracle twice and sends the outputs $y_1$ and $y_2$ as a signal to the decoder. The decoder $\mathsf{Z}$, given $a = \kappa$, computes $x_i = \phi_\kappa^{-1}(y_i)$ (for $i = 1, 2$) and outputs 0 if and only if $x_2 = x_1 + 1$. □

We can also define what it means for a set of subversions to be specious.

**Definition 3** (Specious subversions)**.** Given an efficiently sampleable distribution $\mathsf{D}$ with outputs of the form $(\widetilde{\mathsf{C}}_1, \ldots, \widetilde{\mathsf{C}}_m, a) \leftarrow \mathsf{D}$, we let $\mathsf{D}_i$ be the distribution sampling $(\widetilde{\mathsf{C}}_1, \ldots, \widetilde{\mathsf{C}}_m, a) \leftarrow \mathsf{D}$ and then outputting $(\widetilde{\mathsf{C}}_i, (i, a))$. We say that $\mathsf{D}$ is specious if each $\mathsf{D}_i$ is specious.

We now define the notion of a specious corruption. In this paper, we assume that all specious corruptions are static.

**Definition 4** (Specious corruption)**.** We say that a party accepts specious corruptions if, whenever it gets input $(\textsc{Specious}, \widetilde{\mathsf{C}})$ from the adversary, it replaces its code by $\widetilde{\mathsf{C}}$. If the input $(\textsc{Specious}, \widetilde{\mathsf{C}})$ is not the first one received by the party, then it ignores it. We say that an environment $\mathcal{E}$ prepares specious corruptions if it operates as follows. First, it writes $(\textsc{Specious}, \mathsf{D})$ on a special tape, where $\mathsf{D}$ is specious. Then, it samples $(\widetilde{\mathsf{C}}_1, \ldots, \widetilde{\mathsf{C}}_m, a) \leftarrow \mathsf{D}$ and writes this on the special tape too. Finally, it inputs $(\textsc{Specious}, \widetilde{\mathsf{C}}_1, \ldots, \widetilde{\mathsf{C}}_m)$ to the adversary. The above has to be done on the first activation, before any other communication with protocols or the adversary. We call this a specious environment.

In case of emulation with respect to the dummy adversary, we further require that if the environment instructs the dummy adversary to input $(\textsc{Specious}, \widetilde{\mathsf{C}})$ to a party, then $\widetilde{\mathsf{C}}$ is from the list in $(\textsc{Specious}, \widetilde{\mathsf{C}}_1, \ldots, \widetilde{\mathsf{C}}_m)$. We say that an adversary interacting with a specious environment does specious corruptions if whenever the adversary inputs $(\textsc{Specious}, \widetilde{\mathsf{C}})$ to a party, then $\widetilde{\mathsf{C}}$ is from the list $(\textsc{Specious}, \widetilde{\mathsf{C}}_1, \ldots, \widetilde{\mathsf{C}}_m)$ received from the specious environment. We call such an adversary specious. In particular, an adversary which never inputs $(\textsc{Specious}, \widetilde{\mathsf{C}})$ to any party is specious. We also call an environment specious if it does not write $(\textsc{Specious}, \mathsf{D})$ on a special tape as the first thing, but in this case we require that it does not input anything of the form $(\textsc{Specious}, \widetilde{\mathsf{C}}_1, \ldots, \widetilde{\mathsf{C}}_m)$ to the adversary, and that it never instructs the dummy adversary to input $(\textsc{Specious}, \widetilde{\mathsf{C}})$ to any party.

In addition we require that specious environments and adversaries only do static corruptions and that all corruptions are of the form.

- Core Malicious and firewall Malicious.
- Core Honest and firewall SemiHonest.
- Core Specious and firewall Honest.
- Core Honest and firewall Malicious.

We assume that all cores accept specious corruptions, and no other parties accept specious corruptions.

We add a few comments to the definition. First, let us explain why we only require security for the above four corruption patterns. Of all the corruption patterns shown in Table 1 giving rise to a Malicious party, the one with core Malicious and firewall Malicious gives the adversary strictly more power than any of the other ones, so we only ask for simulation of that case. Similarly, of the 3 corruption patterns giving rise to an Honest party, the ones with the core Honest and Specious and the firewall SemiHonest and Honest respectively are different, as neither gives powers to the adversary which are a subset of the other, so we ask for simulation of both. The remaining case of Honest core and Honest firewall we can drop, as it is a special case of the Honest core and SemiHonest firewall. The only corruption pattern giving rise to an Isolate party is when the core is Honest and the firewall is Malicious; we therefore ask to simulate this case too.

Second, note that it might look odd that we ask the environment to sample the subversion $\widetilde{\mathsf{C}}_i$. Could we not just ask that, when it inputs $(\textsc{Specious}, \widetilde{\mathsf{C}}_i)$ to a core, then $\widetilde{\mathsf{C}}_i$ is specious? It

| Core $\mathsf{C}$ | Firewall $\mathsf{F}$ | Party $\mathsf{P}$ in $\mathcal{F}$ |
|---|---|---|
| Honest | Honest | Honest |
| Honest | SemiHonest | Honest |
| Specious | Honest | Honest |
| Honest | Malicious | Isolate |
| Specious | SemiHonest | Malicious |
| Specious | Malicious | Malicious |
| Malicious | Honest | Malicious |
| Malicious | SemiHonest | Malicious |
| Malicious | Malicious | Malicious |

Table 1: Corruption patterns for cores and firewalls in our model, and their translation in the ideal world. The highlighted rows are the cases that one needs to consider when proving security using our framework.

turns out that this would give a trivial notion of specious corruption. Recall that in the notion of specious, we quantify over all tests. If we first fix $\widetilde{\mathsf{C}}$, and then quantify over all tests when defining that it is specious, then the universal quantifier could be used to guess random values shared between $\widetilde{\mathsf{C}}$ and the adversary, like the key $\kappa$ used in Lemma 1 (demonstrating that a specious subversion can still be signaling). Therefore, a single $\widetilde{\mathsf{C}}$ specious subversion cannot be signalling. Hence, asking for a specific subversion to be specious would make the notion of specious corruption trivial. By instead asking that a distribution $\mathsf{D}$ is specious, we can allow $\widetilde{\mathsf{C}}$ and the adversary to sample joint randomness (like a secret key $\kappa$) after the test $\mathsf{T}$ has already been quantified. Namely, recall that in the test game we first fix a $\mathsf{T}$, and only then do we sample $\mathsf{D}$. This allows specious corruptions which can still signal to the adversary, as demonstrated above. The reason why we ask the environment to sample $\mathsf{D}$ and not the adversary has to do with UC composition, which we return to later.

### 2.4 Sanitizing Protocols Implementing Regular Ideal Functionalities

For illustration, we first describe how to implement a regular ideal functionality given a sanitizing ideal functionality. Later, we cover the case of implementing a sanitizing ideal functionality given a sanitizing ideal functionality, see Fig. 1.

Consider a sanitizing protocol $\Pi$, using a sanitizable ideal functionality $\mathcal{G}$, that implements a regular ideal functionality $\mathcal{F}$ with $n$ parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$. By regular, we mean that $\mathcal{F}$ itself does not have a sanitation interface. Note that it makes perfect sense for a sanitizing protocol $\Pi$, using a sanitizable ideal functionality $\mathcal{G}$, to implement a regular ideal functionality. The firewall is an aspect of the implementation $\Pi$ and the sanitizable hybrid ideal functionality $\mathcal{G}$. In particular, this aspect could be completely hidden by the implementation of $\Pi$. However, typically the behavior when the firewall is honest and corrupted is not the same. A corrupted firewall can isolate the core by not doing its job. We therefore call a party $\mathsf{P}_i$ where $\mathsf{C}_i$ is honest and $\mathsf{F}_i$ is corrupt an "isolated" party. We insist that if $\mathsf{C}_i$ is specious and $\mathsf{F}_i$ is honest, then it is as if $\mathsf{P}_i$ is honest. Hence, $\mathcal{F}$ should behave as if $\mathsf{P}_i$ is honest. We would therefore like the behavior of $\mathcal{F}$ to depend only on whether $\mathsf{P}_i$ is honest, isolated, or corrupt. To add some structure to this, we introduce the notion of a wrapped ideal functionality and a wrapper.

A wrapped ideal functionality $\mathcal{F}$ should only talk to parties $\mathsf{P}_i$. The wrapper $\mathsf{Wrap}$ will talk to a core $\mathsf{C}_i$ and a firewall $\mathsf{F}_i$. The wrapper runs $\mathcal{F}$ internally, and we write $\mathsf{Wrap}(\mathcal{F})$. The inputs to and from $\mathsf{C}_i$ on $\mathsf{Wrap}(\mathcal{F})$ are forwarded to the interface for $\mathsf{P}_i$ on $\mathcal{F}$. The only job of $\mathsf{Wrap}$ is to introduce the same parties as in the protocol and translate corruptions of $\mathsf{C}_i$ and $\mathsf{F}_i$ into corruptions on $\mathsf{P}_i$. We say that parties $\mathsf{P}_i$ in an ideal execution with $\mathcal{F}$ can be Honest, MALICIOUS or ISOLATE. The wrapped ideal functionality $\mathsf{Wrap}(\mathcal{F})$ translates corruptions using the following *standard corruption translation table.*

**Honest:** If $\mathsf{C}_i$ is HONEST and $\mathsf{F}_i$ HONEST, let $\mathsf{P}_i$ be HONEST on $\mathcal{F}$.

**Malicious:** If $\mathsf{C}_i$ is MALICIOUS, corrupt $\mathsf{P}_i$ as MALICIOUS on $\mathcal{F}$.

**Isolated:** If $\mathsf{C}_i$ is HONEST and $\mathsf{F}_i$ is MALICIOUS, corrupt $\mathsf{P}_i$ as ISOLATE on $\mathcal{F}$.

**Sanitation:** If $\mathsf{C}_i$ is SPECIOUS and $\mathsf{F}_i$ is HONEST, let $\mathsf{P}_i$ be HONEST on $\mathcal{F}$.

**No Secrets:** If $\mathsf{C}_i$ is HONEST and $\mathsf{F}_i$ is SEMIHONEST, let $\mathsf{P}_i$ be HONEST on $\mathcal{F}$.

We discuss the five cases next. The **Honest** and **Malicious** cases are straightforward; if both the core and the firewall are honest, then treat $\mathsf{P}_i$ as an honest party on $\mathcal{F}$. Similarly, if the core is malicious, then treat $\mathsf{P}_i$ as a malicious party on $\mathcal{F}$. The **Isolated** case corresponds to the situation where the core is honest and the firewall is corrupted, and thus the firewall is isolating the core from the network. This will typically correspond to a corrupted party. However, in some cases, some partial security might be obtainable, like the inputs of the core being kept secret. We therefore allow an ISOLATE corruption as an explicit type of corruption. The standard behavior of $\mathcal{F}$ on an ISOLATE corruption is to do a MALICIOUS corruption of $\mathsf{P}_i$ in $\mathcal{F}$.

The **Sanitation** case essentially says that the job of the firewall is to turn a specious core into an honest core. This, in particular, means that the firewall should remove any signaling. We add the **No Secrets** case to avoid trivial solutions where the firewall is keeping, *e.g.*, secret keys used in the protocol. We want secret keys to reside in the core, and that firewalls only sanitize communication of the core. We also do not want that the core just hands the inputs to the firewall and lets it run the protocol. A simple way to model this is to require that the protocol should tolerate a semi-honest corruption of the firewall when the core is honest. We do not require that we can tolerate a specious core and a semi-honest firewall. Removing signaling from a core will typically require randomizing some of the communication. For this, the firewall needs to be able to make secret random choices. Note that, with this modeling, a core and a firewall can be seen as a two-party implementation of the honest party, where one can tolerate either a specious corruption of the core or a semi-honest corruption of the firewall.

**Definition 5** (Wrapped subversion-resilient UC security)**.** Let $\mathcal{F}$ be an ideal functionality for $n$ parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Let $\Pi$ be a sanitizing protocol with $n$ cores $\mathsf{C}_1, \ldots, \mathsf{C}_n$ and $n$ firewalls $\mathsf{F}_1, \ldots, \mathsf{F}_n$. Let $\mathcal{G}$ be a sanitizable ideal functionality which can be used by $\Pi$ as in Fig. 1. We say that $\Pi$ wsrUC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if $\Pi$ UC-realizes $\mathsf{Wrap}(\mathcal{F})$ in the $\mathcal{G}$-hybrid model with the restriction that we only quantify over specious environments and specious adversaries.

The typical behavior of a sanitizing ideal functionality is that, when it receives a message from the core, it will output the received message to the firewall, or output some partial information about the message to the firewall. Later, it will receive some new message or sanitation instruction from the firewall. Given this, it constructs the actual information to pass to the core functionality of $\mathcal{G}$. This might later end up at a firewall of another party, and after sanitation end up at the core of that party. The latter is illustrated in Fig. 1, and an example is given below. Note that this is not a formal requirement, but just a description of idiomatic use of sanitation to give an intuition on the use of the model.

To illustrate the use of sanitizable ideal functionalities, we specify an ideal functionality $\mathcal{F}_{\mathsf{SAT}}$ for sanitizable authenticated communication. The communication between cores goes via the firewall which might change the messages. Note that firewalls can be sure which other firewall they talk to, but corrupted firewalls can lie to their local core about who sent a message. In

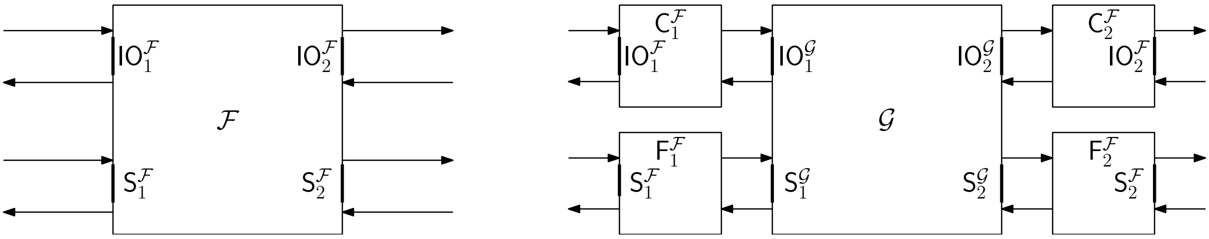

$\mathsf{IO}_1^{\mathcal{F}}$
$\mathsf{IO}_2^{\mathcal{F}}$
$\mathcal{F}$
$\mathsf{S}_1^{\mathcal{F}}$
$\mathsf{S}_2^{\mathcal{F}}$
$\mathsf{C}_1^{\mathcal{F}}$
$\mathsf{IO}_1^{\mathcal{F}}$
$\mathsf{IO}_1^{\mathcal{G}}$
$\mathsf{IO}_2^{\mathcal{G}}$
$\mathsf{C}_2^{\mathcal{F}}$
$\mathsf{IO}_2^{\mathcal{F}}$
$\mathcal{G}$
$\mathsf{F}_1^{\mathcal{F}}$
$\mathsf{S}_1^{\mathcal{F}}$
$\mathsf{S}_1^{\mathcal{G}}$
$\mathsf{S}_2^{\mathcal{G}}$
$\mathsf{F}_2^{\mathcal{F}}$
$\mathsf{S}_2^{\mathcal{F}}$

Figure 2: Implementing $\mathcal{F}$ via protocol $\Pi = (\mathsf{C}^{\mathcal{F}}, \mathsf{F}^{\mathcal{F}})$ using $\mathcal{G}$.

fact, they can pretend a message arrived out of the blue. We also equip $\mathcal{F}_{\mathsf{SAT}}$ with the possibility for distributing setup, as this is needed in some of our protocols. We assume a setup generator $\mathsf{Setup}$ which samples the setup and gives each party their corresponding value. The firewalls also get a value. This, *e.g.*, allows to assume that the firewalls know a CRS. Since we do not want firewalls to keep secrets, we leak their setup values to the adversary. This would not be a problem if the setup values is a CRS.

**Functionality $\mathcal{F}_{\mathsf{SAT}}$**

- Initially sample $((v_1, w_1), \ldots, (v_n, w_n)) \leftarrow \mathsf{Setup}()$ and output $v_i$ on $\mathsf{IO}_i$ and $w_i$ on $\mathsf{S}_i$. Leak $w_i$ to the adversary.
- On input $(\textsc{Send}, a, \mathsf{P}_j)$ on $\mathsf{IO}_i$, output $(\textsc{Send}, a, \mathsf{P}_j)$ on $\mathsf{S}_i$. To keep the description simple we assume honest parties sends the same $a$ at most once. Adding fresh message identifiers can be used for this in an implementation.
- On input $(\textsc{Send}, b, \mathsf{P}_k)$ on $\mathsf{S}_i$, leak $(\textsc{Send}, \mathsf{P}_i, b, \mathsf{P}_k)$ to the adversary and store $(\textsc{Send}, \mathsf{P}_i, b, \mathsf{P}_k)$.
- On input $(\textsc{Deliver}, (\textsc{Send}, \mathsf{P}_i, b, \mathsf{P}_k))$ from the adversary, where $(\textsc{Send}, \mathsf{P}_i, b, \mathsf{P}_k)$ is stored, delete this tuple and output $(\textsc{Receive}, \mathsf{P}_i, b)$ on $\mathsf{S}_k$.
- On input $(\textsc{Receive}, \mathsf{P}_m, c)$ on $\mathsf{S}_k$, output $(\textsc{Receive}, \mathsf{P}_m, c)$ on $\mathsf{IO}_i$.

*Remark* 1 (on $\mathcal{F}_{\mathsf{SAT}}$). We note that all protocols in this work, even if not explicitly stated, are described in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model. Moreover, whenever we say that the core sends a message to the firewall (or vice-versa) we actually mean that they communicate using $\mathcal{F}_{\mathsf{SAT}}$.

## 2.5 General Case

We now turn our attention to implementing sanitizable ideal functionalities. When a protocol $\Pi$ implements a sanitizable ideal functionality, we call $\Pi$ a sanitizable protocol. Notice the crucial difference between being a *sanitizable* protocol and a *sanitizing* protocol. A sanitizable protocol $\Pi$ *implements* the sanitization interface $\mathsf{S}_i$ of $\mathcal{F}$. Whereas a sanitizing protocol $\Pi$ would have a firewall *using* the sanitization interface $\mathsf{S}_i$ of $\mathcal{G}$.

When implementing a sanitizable ideal functionality $\mathcal{F}$, the protocol should implement the sanitation interface $\mathsf{S}^{\mathcal{F}}$ for $\mathsf{F}$. This means that the protocol will be of the form $\Pi = (\mathsf{IO}, \mathsf{S})$ where $\mathsf{IO} = (\mathsf{IO}_1, \ldots, \mathsf{IO}_n)$ and $\mathsf{S} = (\mathsf{S}_1, \ldots, \mathsf{S}_n)$. Notice that $\mathsf{C}_i$ and $\mathsf{F}_i$ formally are separate parties, so they cannot talk directly.

It is natural that it is the firewall of the implementation $\Pi = (\mathsf{IO}, \mathsf{S})$ which handles this. The firewall has access to the sanitation interface of $\mathcal{G}$, which it can use to sanitize $\Pi$. This means that $\mathsf{F}$ gets what could look like a double role now. First, it sanitizes $\Pi$ using $\mathsf{S}^{\mathcal{G}}$. Second, it has to implement the sanitation interface $\mathsf{S}^{\mathcal{F}}$ of $\Pi$ (matching that of $\mathcal{F}$). Note, however, that this is in fact the same job. The sanitation interface $\mathsf{S}^{\mathcal{F}}$ of $\Pi$ is used to specify how $\Pi$ should be

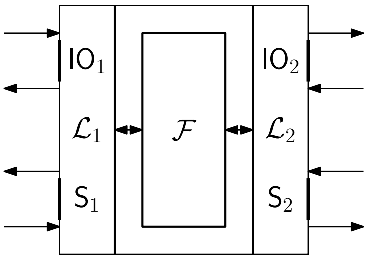
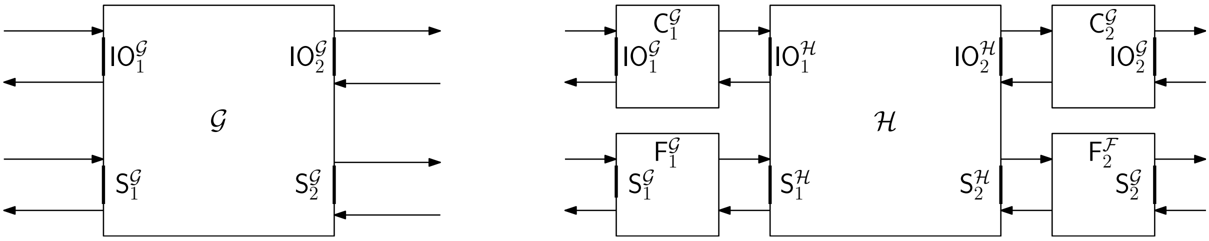


Figure 3: The wrapper $\mathsf{Wrap}(\mathcal{F}, \mathcal{L}_1, \ldots, \mathcal{L}_n))$.



$\mathsf{IO}_1^{\mathcal{G}}$
$\mathsf{IO}_2^{\mathcal{G}}$
$\mathcal{G}$
$\mathsf{S}_1^{\mathcal{G}}$
$\mathsf{S}_2^{\mathcal{G}}$
$\mathsf{C}_1^{\mathcal{G}}$
$\mathsf{IO}_1^{\mathcal{G}}$
$\mathsf{F}_1^{\mathcal{G}}$
$\mathsf{S}_1^{\mathcal{G}}$
$\mathsf{IO}_1^{\mathcal{H}}$
$\mathsf{IO}_2^{\mathcal{H}}$
$\mathcal{H}$
$\mathsf{S}_1^{\mathcal{H}}$
$\mathsf{S}_2^{\mathcal{H}}$
$\mathsf{C}_2^{\mathcal{G}}$
$\mathsf{IO}_2^{\mathcal{G}}$
$\mathsf{F}_2^{\mathcal{F}}$
$\mathsf{S}_2^{\mathcal{G}}$

Figure 4: Implementing $\mathcal{G}$ via protocol $\Gamma = (\mathsf{C}^{\mathcal{G}}, \mathsf{F}^{\mathcal{G}})$ using $\mathcal{H}$.

sanitized. It is natural that $\mathsf{F}^{\mathcal{F}}$ needs to knows this specification. It then uses $\mathsf{S}^{\mathcal{G}}$ to implement the desired sanitation. This is illustrated in Fig. 2.

When defining security of a protocol implementing a sanitizable ideal functionality, we do not need to use a wrapper as when implementing a normal ideal functionality, as $\mathcal{F}$ already has the same parties as in the protocol. It is however still convenient to use a wrapper to add some structure to how we specify a sanitizable ideal functionality. We assume a central part which does the actual computation, and outer parts which sanitize the inputs from $\mathsf{P}_i$ before they are passed to the central part.

**Definition 6** (Well-formed sanitizing ideal functionality)**.** A well-formed sanitizing ideal functionality consists of an ideal functionality $\mathcal{F}$, called the central part, with an interface $\mathsf{P}_i$ for each party. The interface $\mathsf{P}_i$ can be Honest, Malicious, or Isolate. There are also $n$ outer parts $\mathcal{L}_1, \ldots, \mathcal{L}_n$ where $\mathcal{L}_i$ has an interface $\mathsf{IO}_i$ for the core and $\mathsf{S}_i$ for the firewall. The outer part $\mathcal{L}_i$ can only talk to the central part on $\mathsf{P}_i$ and the outer parts cannot communicate with each other. The interface $\mathsf{IO}_i$ can be Honest, Malicious, or Specious. The interface $\mathsf{S}_i$ can be Honest, Malicious, or SemiHonest. The corruption of $\mathcal{F}.\mathsf{IO}_i$ is computed from that of $\mathcal{L}_i.\mathsf{IO}_i$ and $\mathcal{L}_i.\mathsf{S}_i$ using the standard corruption translation table.

**Definition 7** (Subversion-resilient UC security)**.** Let $\mathcal{F}$ be an ideal functionality for $n$ cores $\mathsf{C}_1^{\mathcal{F}}, \ldots, \mathsf{C}_n^{\mathcal{F}}$ and $n$ firewalls $\mathsf{F}_1^{\mathcal{F}}, \ldots, \mathsf{F}_n^{\mathcal{F}}$, and let $\Pi$ be a sanitizing protocol with $n$ cores $\mathsf{C}_1^{\mathcal{F}}, \ldots, \mathsf{C}_n^{\mathcal{F}}$ and $n$ firewalls $\mathsf{F}_1^{\mathcal{F}}, \ldots, \mathsf{F}_n^{\mathcal{F}}$. Let $\mathcal{G}$ be a sanitizable ideal functionality which can be used by $\Pi$ as in Fig. 2. We say that $\Pi$ srUC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if $\mathcal{F}$ can be written as a well-formed sanitizing ideal functionality, and $\Pi$ UC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model with the restriction that we only quantify over specious environments and specious adversaries.

## 2.6 Composition

We now address composition. In Fig. 2, we illustrate implementing $\mathcal{F}$ in the $\mathcal{G}$-hybrid model. Similarly, in Fig. 4, we implement $\mathcal{G}$ given $\mathcal{H}$. In Fig. 5, we illustrate the effect of composition. We

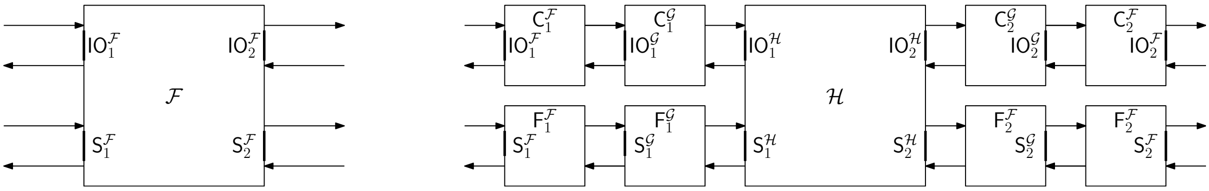

$\mathsf{IO}_1^{\mathcal{F}}$
$\mathsf{IO}_2^{\mathcal{F}}$
$\mathcal{F}$
$\mathsf{S}_1^{\mathcal{F}}$
$\mathsf{S}_2^{\mathcal{F}}$
$\mathsf{C}_1^{\mathcal{F}}$
$\mathsf{IO}_1^{\mathcal{F}}$
$\mathsf{C}_1^{\mathcal{G}}$
$\mathsf{IO}_1^{\mathcal{G}}$
$\mathsf{IO}_1^{\mathcal{H}}$
$\mathsf{IO}_2^{\mathcal{H}}$
$\mathsf{C}_2^{\mathcal{G}}$
$\mathsf{IO}_2^{\mathcal{G}}$
$\mathsf{C}_2^{\mathcal{F}}$
$\mathsf{IO}_2^{\mathcal{F}}$
$\mathcal{H}$
$\mathsf{F}_1^{\mathcal{F}}$
$\mathsf{S}_1^{\mathcal{F}}$
$\mathsf{F}_1^{\mathcal{G}}$
$\mathsf{S}_1^{\mathcal{G}}$
$\mathsf{S}_1^{\mathcal{H}}$
$\mathsf{S}_2^{\mathcal{H}}$
$\mathsf{F}_2^{\mathcal{F}}$
$\mathsf{S}_2^{\mathcal{G}}$
$\mathsf{F}_2^{\mathcal{F}}$
$\mathsf{S}_2^{\mathcal{F}}$

Figure 5: Implementing $\mathcal{F}$ via protocol $\Pi^{\mathcal{G}\to\Gamma}$ using $\mathcal{H}$.

can let $\mathsf{C}_i = \mathsf{C}_i^{\mathcal{F}} \circ \mathsf{C}_i^{\mathcal{G}}$ and $\mathsf{F}_i = \mathsf{F}_i^{\mathcal{F}} \circ \mathsf{F}_i^{\mathcal{G}}$. Then, we again have a sanitizing protocol $\Pi^{\mathcal{G}\to\Gamma} = (\mathsf{C}, \mathsf{F})$. For composition to work, we need that specious corruptions respect the composition of a core.

**Definition 8** (Specious corruption of a composed core)**.** We say that an adversary does a specious corruption of a composed core $\mathsf{C}_i = \mathsf{C}_i^{\mathcal{F}} \circ \mathsf{C}_i^{\mathcal{G}}$ if it inputs $(\textsc{Specious}, \widetilde{\mathsf{C}}_i^{\mathcal{F}}, \widetilde{\mathsf{C}}_i^{\mathcal{G}})$, where both $\mathsf{C}_i^{\mathcal{F}}$ and $\mathsf{C}_i^{\mathcal{G}}$ are specious. In response $\mathsf{C}_i^{\mathcal{F}}$ replaces its code with $\widetilde{\mathsf{C}}_i^{\mathcal{F}}$, and $\mathsf{C}_i^{\mathcal{G}}$ replaces its code with $\widetilde{\mathsf{C}}_i^{\mathcal{G}}$.

Note that one could imagine a specious corruption of a composed core $\mathsf{C}_i$ which could not be written as the composition of specious subversions $\widetilde{\mathsf{C}}_i^{\mathcal{F}}$ and $\widetilde{\mathsf{C}}_i^{\mathcal{G}}$.

**Theorem 1** (srUC Composition)**.** *Let $\mathcal{F}$ and $\mathcal{G}$ be ideal functionalities, and let $\Pi$ and $\Gamma$ be protocols. Assume that all are subroutine respecting and subroutine exposing as defined in [Can00]. If $\Pi$ srUC-realizes $\mathcal{F}$, and $\Gamma$ srUC-realizes $\mathcal{G}$, then $\Pi^{\mathcal{G}\to\Gamma}$ srUC-realizes $\mathcal{F}$.*

*Proof sketch.* Recall that UC composition gives us that if $\Pi$ UC-realizes $\mathcal{F}$ and $\Gamma$ UC-realizes $\mathcal{G}$, then $\Pi^{\mathcal{G}\to\Gamma}$ UC-realizes $\mathcal{F}$, which would imply that $\Pi^{\mathcal{G}\to\Gamma}$ srUC-realizes $\mathcal{F}$. However, we cannot directly use this result, as in srUC we only quantify over specious environments and specious adversaries. Thus, we cannot conclude that $\Pi$ UC-realizes $\mathcal{F}$ from the fact that $\Pi$ srUC-realizes $\mathcal{F}$. We therefore have to white-box inspect the proof of the UC theorem to ensure that it still goes through for specious environments and adversaries. We do that next.[1]

Recall that the crucial point in the proof of general UC composition [Can00, Theorem 22] is proving that if two protocols $\Phi$ and $\Pi$ are such that $\Phi$ UC-emulates $\Pi$ and $\Psi$ is a protocol using $\Phi$, and $\Psi^{\Phi\to\Pi}$ is $\Psi$ with $\Phi$ replaced by $\Pi$, then $\Psi$ UC-emulates $\Psi^{\Phi\to\Pi}$. The crucial proof step is to construct from an environment $\mathcal{E}$ talking to a dummy adversary $\mathcal{A}$, and attacking $\Psi$ or $\Psi^{\Phi\to\Pi}$, a new environment $\mathcal{E}_\Pi$ which is an environment for a single instance of $\Phi$ or $\Pi$, as illustrated in [Can00, Fig. 9]. Environment $\mathcal{E}_\Pi$ attacking $\Phi$ will have the same effect as $\mathcal{E}$ attacking $\Psi$. Environment $\mathcal{E}_\Pi$ attacking $\Pi$ will have the same effect as $\mathcal{E}$ attacking $\Psi^{\Phi\to\Pi}$. One can then appeal to the fact that $\Phi$ UC-emulates $\Pi$ to prove that $\Psi$ UC-emulates $\Psi^{\Phi\to\Pi}$. When proving srUC security, it is crucial in the last step that if $\mathcal{E}$ is specious then $\mathcal{E}_\Pi$ is specious, as we only quantify over specious environments.

To see that $\mathcal{E}_\Pi$ is specious note that it is constructed blackbox from $\mathcal{E}$, and that it runs $\mathcal{E}$ as the first thing. Therefore, it will correctly sample and write $(\widetilde{\mathsf{C}}_1, \ldots, \widetilde{\mathsf{C}}_m, a)$ to a special tape as the first thing it does. Furthermore, since $\mathcal{E}$ runs with a dummy adversary, all specious corruptions are instructed by $\mathcal{E}$. The environment $\mathcal{E}_\Pi$ will by construction only do specious corruptions which $\mathcal{E}$ instructed. Therefore, whenever $\mathcal{E}_\Pi$ does a corruption $(\textsc{Specious}, \widetilde{\mathsf{C}})$, then $\widetilde{\mathsf{C}}$ is from the list in $(\widetilde{\mathsf{C}}_1, \ldots, \widetilde{\mathsf{C}}_m, a)$, as required. Hence, $\mathcal{E}_\Pi$ is again specious.

Since the UC composition proof [Can00, Theorem 22] assumes a dummy adversary, composition also depends on universality of dummy adversaries [Can00, Claim 11] which allows to turn an environment $\mathcal{E}$ and an adversary $\mathcal{A}$ into an equivalent new adversary $\mathcal{E}_{\mathcal{A}}$ which talks

[1] The remainder of the proof requires some familiarity with the proof of the UC composition theorem [Can00].

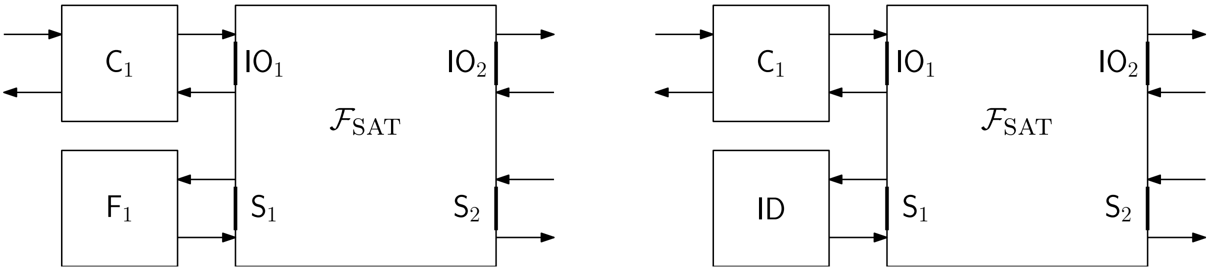


Figure 6: A core with its matching firewall or with the identity firewall.

to a dummy adversary. A dummy adversary is one which just acts as a channel between the environment and the protocol. The environment $\mathcal{E}_{\mathcal{A}}$ just runs $\mathcal{E}$ and $\mathcal{A}$, and instructs the dummy adversary to forward the messages implementing the attack of $\mathcal{A}$. It is easy to see that, if $\mathcal{E}$ is a specious environment and $\mathcal{A}$ is a specious adversary, then $\mathcal{E}_{\mathcal{A}}$ is a specious environment. □

Note that if, *e.g.*, $\mathcal{G}$ in the composition is well-formed and therefore wrapped, then it is the wrapped functionality which is considered at all places. Therefore, in Fig. 4 the ideal functionality $\mathcal{G}$ being implemented will be the wrapped ideal functionality, and in Fig. 2 the hybrid ideal functionality $\mathcal{G}$ being used would again be the wrapped one. There is no notion of "opening up the wrapping" during composition. If $\mathcal{F}$ is a regular ideal functionality then $\mathsf{Wrap}(\mathcal{F})$ can be written as a well-formed sanitizing ideal functionality. Therefore wsrUC security relative to $\mathcal{F}$ implies srUC security relative to $\mathsf{Wrap}(\mathcal{F})$. During composition it would be $\mathsf{Wrap}(\mathcal{F})$ which is used as a hybrid functionality. This is basically the same as having $\mathcal{F}$ under the standard corruption translation.

## 2.7 Computational Transparency

A central notion in the study of reverse firewalls is the notion of transparency. The firewall is only supposed to modify the behavior of a subverted core. If the firewall is attached to an honest core, it must not change the behavior of the core. We define transparency in line with [MS15], namely, an honest core without a firewall attached should be indistinguishable from an honest core with a firewall attached.

Notice that this does not make sense if the party is implementing a sanitizable ideal functionality, like in Fig. 2. Without a firewall $\mathsf{F}_1^{\mathcal{F}}$, no entity would implement the interface $\mathsf{S}_1^{\mathcal{F}}$, which would make a core without a firewall trivially distinguishishable from a core with a firewall. Presumably, the interface $\mathsf{S}_1^{\mathcal{F}}$ is present because different inputs on this interface will give different behaviors. We therefore only define transparency of firewalls implementing a regular ideal functionality, as in Fig. 1. Note also that if $\mathcal{G}$ in Fig. 1 has a complex interaction with $\mathsf{F}_i$, then an execution without $\mathsf{F}_i$ might not make sense. Therefore, we additionally only consider transparency in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model. In this model we can let $\mathsf{F}_i$ be an *identity firewall* which does not modify the communication. This has the desired notion of no firewall being present.

**Definition 9** (Transparency). Let $(\mathsf{C}_i, \mathsf{F}_i)$ be a party for the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model. Let $\Pi_i$ be the protocol for the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model where party number $i$ is $(\mathsf{C}_i, \mathsf{F}_i)$, and all other parties are dummy parties. Let $\mathsf{ID}$ be the firewall which always outputs any message it receives as input. Let $\Pi_i'$ be the protocol for the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model where party number $i$ is $(\mathsf{C}_i, \mathsf{ID})$, and all other parties are dummy parties. These two protocols are illustrated in Fig. 6. We say that $\mathsf{F}_i$ is

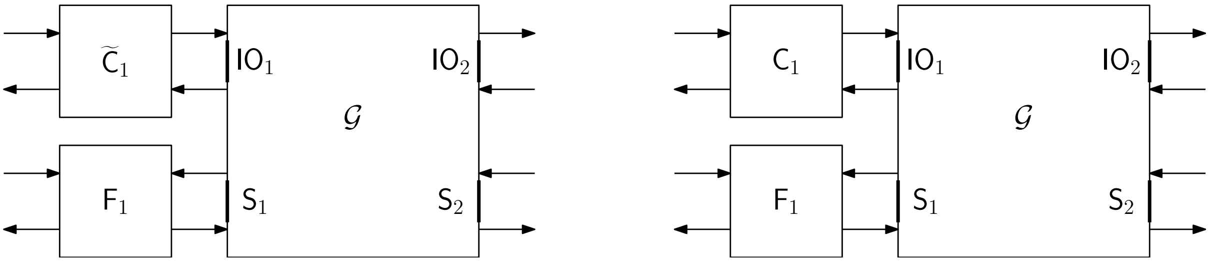


Figure 7: An honest core with its matching firewall or a specious core with the same firewall.

*computationally transparent* if, for all poly-time environments $\mathcal{E}$ which do not corrupt $\mathsf{C}_i$ or $\mathsf{F}_i/\mathsf{ID}$, it holds that $\textsc{Exec}_{\mathcal{E},\Pi_i,\mathcal{A}} \approx \textsc{Exec}_{\mathcal{E},\Pi'_i,\mathcal{A}}$, where $\mathcal{A}$ is the dummy adversary.

### 2.8 Strong Sanitation

Another central notion in the study of reverse firewalls is the notion that we call sanitation. Namely, if you hide a specious core behind a firewall, then it looks like an honest core behind a firewall. So far, we have defined this implicitly by saying that a specious corruption of a core plus an honest firewall should be simulatable by having access to an honest party on the ideal functionality being implemented. This actually does not imply that the network cannot distinguish between a specious core or an honest core behind the firewall. It only says that the effect of a specious core behind a firewall are not dire enough that you cannot simulate given an honest party in the ideal world.

In this section, we give a game-based definition of sanitation capturing the stronger notion that, behind a firewall, a specious core looks like an honest core. Recall that a core $\mathsf{C}_i$ is capable of receiving a specious corruption $(\textsc{Specious}, \widetilde{\mathsf{C}})$ from the environment, in which case it replaces its code by $\widetilde{\mathsf{C}}$. For such a core, let $\widehat{\mathsf{C}}$ be the *incorruptible core* which when it receives a specious corruption $(\textsc{Specious}, \widetilde{\mathsf{C}})$ will ignore it and keep running the code of $\mathsf{C}$.

**Definition 10** (Strong sanitation)**.** Let $(\mathsf{C}_i, \mathsf{F}_i)$ by a party for the $\mathcal{G}$-hybrid model. Let $\widehat{\mathsf{C}}_i$ be the corresponding incorruptible core. Let $\Pi_i$ be the protocol for the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model where party number $i$ is $(\mathsf{C}_i, \mathsf{F}_i)$, and all other parties are dummy parties. Let $\Pi'_i$ be the protocol for the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model where party number $i$ is $(\widehat{\mathsf{C}}_i, \mathsf{F}_i)$, and all other parties are dummy parties. Note that if the environment does a $(\textsc{Specious}, \widetilde{\mathsf{C}})$ corruption of core number $i$, then in $\Pi_i$ core $i$ will run $\widetilde{\mathsf{C}}$, whereas in $\Pi'_i$ it will run $\mathsf{C}_i$. These two outcomes are illustrated in Fig. 7. We say that $\mathsf{F}_i$ is *strongly sanitising* if, for all poly-time environments $\mathcal{E}$ which do not corrupt $\mathsf{F}_i$, but which are allowed a specious corruption of the core, it holds that $\textsc{Exec}_{\mathcal{E},\Pi_i,\mathcal{A}} \approx \textsc{Exec}_{\mathcal{E},\Pi'_i,\mathcal{A}}$, where $\mathcal{A}$ is the dummy adversary.

It is easy to see that the definition is equivalent to requiring that, for all poly-time environments $\mathcal{E}$ which do not corrupt $\mathsf{C}_i$ or $\mathsf{F}_i/\mathsf{ID}$, it holds that $\textsc{Exec}_{\mathcal{E},\Pi_i,\mathcal{A}} \approx \textsc{Exec}_{\mathcal{E},\Pi'_i,\mathcal{A}}$, where $\mathcal{A}$ is the dummy adversary.

**Lemma 2.** *Consider a protocol $\Pi$ where for all parties $(\mathsf{C}_i, \mathsf{F}_i)$ it holds that $\mathsf{F}_i$ has strong sanitation. Then it is enough to prove security for these cases:*

- *Core* Malicious *and firewall* Malicious.
- *Core* Honest *and firewall* SemiHonest.
- *Core* Honest *and firewall* Malicious.

*If in addition we assume the standard corruption behavior for* Isolate*, it is enough to prove the cases:*

- *Core* Malicious *and firewall* Malicious*.*
- *Core* Honest *and firewall* SemiHonest*.*

*If in addition the protocol* $\Pi$ *is for the* $\mathcal{F}_{\mathsf{SAT}}$*-hybrid model and has computational transparency, then it is enough to prove the case:*

- *Core* Malicious *and firewall* Malicious*.*
- *Core* Honest *and firewall* Honest*.*

*Proof.* We prove the first claim. Note that relative to Definition 7 we removed the case with the core Specious and the firewall Honest. We show that this reduces to the case with core Honest and the firewall Honest. First replace each $\mathsf{C}_i$ by $\widehat{\mathsf{C}}_i$. This cannot be noticed due to strong sanitation. Then notice that we can replace an environment $\mathcal{E}$ doing specious corruption by $\mathcal{E}'$ which just internally do not pass on $(\text{Specious}, \widetilde{\mathsf{C}})$ to the core. Namely, it does not matter if $\widehat{\mathsf{C}}_i$ ignores the commands or we let $\mathcal{E}'$ do it. Then, we can replace $\widehat{\mathsf{C}}_i$ by $\mathsf{C}_i$ as there are no commands to ignore. So it is enough to prove security for the core Honest and the firewall Honest. This case follows from the case with the core Specious and the firewall Honest as being honest is a special case of being specious.

The second claim follows from the fact that under standard corruption behavior for Isolate the party $\mathsf{P}_i$ on the ideal functionality is Malicious when the firewall is Malicious. So the simulator has the same power when simulating an honest core and malicious firewall as when simulating a malicious core and a malicious firewall. Then note that being an honest core is a special case of being a malicious core.

In the last claim, we have to prove that assuming computational transparency one does not have to prove the case with the core Honest and the firewall SemiHonest. One can instead prove the case with the core Honest and the firewall Honest. To see this note that, by definition of transparency, we can replace the firewall with the identity firewall $\mathsf{ID}$. For this firewall, an Honest corruption is as powerful as a SemiHonest corruption. This is because the only effect of a semi-honest corruption of $\mathsf{ID}$ is to leak the internal value $w_i$ from the setup and the communication sent via $\mathsf{ID}$. The ideal functionality $\mathcal{F}_{\mathsf{SAT}}$ already leaks that information when $\mathsf{ID}$ is honest. □

# 3 String Commitment

In this section, we show how to build UC string commitments with security in the presence of subversion attacks. In particular, after introducing the sanitizable commitment functionality, we exhibit a non-interactive commitment (with an associated reverse firewall) that UC realizes this functionality in the CRS model, under the DDH assumption.

## 3.1 Sanitizable Commitment Functionality

The sanitazable commitment functionality $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$, which is depicted below, is an extension of the standard functionality for UC commitments [CF01]. Roughly, $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$ allows the core of a party to commit to a $\lambda$-bit string $s_i$; the ideal functionality stores $s_i$ and informs the corresponding firewall that the core has sent a commitment. Hence, via the sanitation interface, the firewall of that party is allowed to forward to the functionality a blinding factor $r_i \in \{0,1\}^\lambda$ that is used to blind $s_i$, yielding a sanitized input $\hat{s}_i = s_i \oplus r_i$. At this point, all other parties are informed by the functionality that a commitment took place. Finally, each party is allowed to open the commitment via the functionality, in which case all other parties learn the sanitized input $\hat{s}_i$.

**Functionality $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$**

The sanitizable string commitment functionality $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$ runs with parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ (each consisting of a core $\mathsf{C}_i$ and a firewall $\mathsf{F}_i$), and an adversary $\mathcal{S}$. The functionality consists of the following communication interfaces for the cores and the firewalls respectively.

**Interface IO**

- Upon receiving a message $(\textsc{Commit}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, s_i)$ from $\mathsf{C}_i$, where $s_i \in \{0,1\}^\lambda$, record the tuple $(\mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, s_i)$ and send the message $(\textsc{Receipt}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i)$ to $\mathsf{F}_i$. Ignore subsequent commands of the form $(\textsc{Commit}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \cdot)$.
- Upon receiving a message $(\textsc{Open}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i)$ from $\mathsf{C}_i$, proceed as follows: If the tuple $(\mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \hat{s}_i)$ is recorded and the message $(\textsc{Blind}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \cdot)$ was sent to $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$, then send the message $(\textsc{Open}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \hat{s}_i)$ to all $\mathsf{C}_{j \neq i}$ and $\mathcal{S}$. Otherwise, do nothing.

**Interface S**

- Upon receiving a message $(\textsc{Blind}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, r_i)$ from $\mathsf{F}_i$, where $r_i \in \{0,1\}^\lambda$, proceed as follows: If the tuple $(\mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, s_i, \cdot)$ is recorded, then modify the tuple to be $(\mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \hat{s}_i = s_i \oplus r_i)$ and send the message $(\textsc{Blinded}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, r_i)$ to $\mathsf{C}_i$, and $(\textsc{Receipt}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i)$ to all $\mathsf{C}_{j \neq i}$ and $\mathcal{S}$; otherwise do nothing. Ignore future commands of the form $(\textsc{Blind}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \cdot)$.

### 3.2 Protocol from DDH

Next, we present a protocol that UC-realizes $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$ in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model. For simplicity, let us first consider the case where there are only two parties. The CRS in our protocol is a tuple $\mathsf{crs} = (g, h, T_1, T_2)$ satisfying the following properties:

- The element $g$ is a generator of a cyclic group $\mathbb{G}$ with prime order $q$, and $h, T_1, T_2 \in \mathbb{G}$. Moreover, the DDH assumption holds in $\mathbb{G}$.[2]
- In the real-world protocol, the tuple $(g, h, T_1, T_2)$ corresponds to a non-DH tuple. Namely, it should be the case that $T_1 = g^x$ and $T_2 = h^{x'}$, for $x \neq x'$.
- In the security proof, the simulator will set the CRS as $(g, h, T_1, T_2)$, where $T_1 = g^x$ and $T_2 = h^x$. By the DDH assumption, this distribution is computationally indistinguishable from the real-world distribution. In addition, the simulator will be given the trapdoor $(x, t)$ for the CRS $\mathsf{crs} = (g, h, T_1, T_2)$, such that $h = g^t$ and $T_1 = g^x$.

As explained in Section 1.3, the above ideas can be generalized to the multiparty setting by using a different CRS for each pair of parties. We proceed below to the formal description of the full protocol.

**Protocol $\widehat{\Pi}_{\mathsf{sCOM}}$ (Sanitizable UC Commitment Protocol)**

The protocol is executed between parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ each consisting of a core $\mathsf{C}_i$ and a firewall $\mathsf{F}_i$. In what follows, let party $\mathsf{P}_j = (\mathsf{C}_j, \mathsf{F}_j)$ be the committer, and all other parties $\mathsf{P}_{k \neq j}$ act as verifiers.

**Public inputs:** Group $\mathbb{G}$ with a generator $g$, field $\mathbb{Z}_q$, and $\mathsf{crs} = (\mathsf{crs}_{j,k})_{j,k \in [n], k \neq j} = (g_{j,k}, h_{j,k}, T_{1,j,k}, T_{2,j,k})_{j,k \in [n], k \neq j}$.

[2] Recall that the DDH assumption states that the distribution ensembles $\{g, h, g^x, h^x : x \leftarrow \mathbb{Z}_q\}$ and $\{g, h, g^x, h^{x'} : x, x' \leftarrow \mathbb{Z}_q\}$ are computationally indistinguishable.

**Private inputs:** The committer (or core) $\mathsf{C}_j$ has an input $s \in \{0,1\}^\lambda$ which we parse as $s = (s[1], \cdots, s[\lambda])$. We will encode each bit $s[i] \in \{0,1\}$ with a value $u[i] \in \{-1,1\}$, so that $u[i] = 1$ if $s[i] = 1$ and $u[i] = -1$ if $s[i] = 0$. The firewall $\mathsf{F}_j$ has an input $r = (r[1], \cdots, r[\lambda]) \in \{0,1\}^\lambda$ (*i.e.*, the blinding factor).

**Commit phase:** For all $i \in [\lambda]$, the core $\mathsf{C}_j$ samples a random $\alpha_{j,k}[i] \leftarrow \mathbb{Z}_q$ and computes the values $B_{j,k}[i] = g_{j,k}^{\alpha_{j,k}[i]} \cdot T_{1,j,k}^{u[i]}$ and $H_{j,k}[i] = h_{j,k}^{\alpha_{j,k}[i]} \cdot T_{2,j,k}^{u[i]}$. Hence, it sends $c_{j,k} = (c_{j,k}[1], \cdots, c_{j,k}[\lambda])$ to the firewall $\mathsf{F}_j$ where $c_{j,k}[i] = (B_{j,k}[i], H_{j,k}[i])$. For all $i \in [\lambda]$, the firewall $\mathsf{F}_j$ picks random $\beta_{j,k} = (\beta_{j,k}[1], \cdots, \beta_{j,k}[\lambda]) \in \mathbb{Z}_q^\lambda$ and does the following:

- If $r[i] = 0$, it lets $\widehat{B}_{j,k}[i] = B_{j,k}[i] \cdot g_{j,k}^{\beta_{j,k}[i]}$ and $\widehat{H}_{j,k}[i] = H_{j,k}[i] \cdot h_{j,k}^{\beta_{j,k}[i]}$;
- Else if $r[i] = 1$, it lets $\widehat{B}_{j,k}[i] = B_{j,k}[i]^{-1} \cdot g_{j,k}^{\beta_{j,k}[i]}$ and $\widehat{H}_{j,k}[i] = H_{j,k}[i]^{-1} \cdot h_{j,k}^{\beta_{j,k}[i]}$.

Hence, $\mathsf{F}_j$ sends $\hat{c}_{j,k} = (\hat{c}_{j,k}[1], \cdots, \hat{c}_{j,k}[\lambda])$ to all other parties $\mathsf{P}_{k \neq j}$, where $\hat{c}_{j,k}[i] = (\widehat{B}_{j,k}[i], \widehat{H}_{j,k}[i])$.

**Opening phase:** The core $\mathsf{C}_j$ sends $(s, \alpha_{j,k})$ to the firewall $\mathsf{F}_j$, where $s \in \{0,1\}^\lambda$ and $\alpha_{j,k} \in \mathbb{Z}_q^\lambda$. Upon receiving $(s, \alpha_{j,k})$ from $\mathsf{C}_j$, the firewall $\mathsf{F}_j$ parses $s = (s[1], \cdots, s[\lambda])$ and $\alpha_{j,k} = (\alpha_{j,k}[1], \cdots, \alpha_{j,k}[\lambda])$. Thus, for all $i \in [\lambda]$, it does the following:

- If $r[i] = 0$, it lets $\hat{s}[i] = s[i]$ and $\hat{\alpha}_{j,k}[i] = \alpha_{j,k}[i] + \beta_{j,k}[i]$;
- Else if $r[i] = 1$, it lets $\hat{s}[i] = -s[i]$ and $\hat{\alpha}_{j,k}[i] = -\alpha_{j,k}[i] + \beta_{j,k}[i]$.

Hence, $\mathsf{F}_j$ sends $(\hat{s}, \hat{\alpha}_{j,k})$ to all other parties $\mathsf{P}_{k \neq j}$, where $\hat{s} = (\hat{s}[1], \cdots, \hat{s}[\lambda])$ and $\hat{\alpha}_{j,k} = (\hat{\alpha}_{j,k}[1], \cdots, \hat{\alpha}_{j,k}[\lambda])$.

**Verification phase:** Upon receiving $(\hat{c}_{j,k}, (\hat{s}, \hat{\alpha}_{j,k}))$ from $\mathsf{P}_j$, each party $\mathsf{P}_{k \neq j}$ parses $\hat{c}_{j,k} = ((\widehat{B}_{j,k}[1], \widehat{H}_{j,k}[1]), \cdots, (\widehat{B}_{j,k}[\lambda], \widehat{H}_{j,k}[\lambda]))$, $\hat{\alpha}_{j,k} = (\hat{\alpha}_{j,k}[1], \cdots, \hat{\alpha}_{j,k}[\lambda])$, and encodes $\hat{s} = (\hat{s}[1], \cdots, \hat{s}[\lambda]) \in \{0,1\}^\lambda$ as $\hat{u} = (\hat{u}[1], \cdots, \hat{u}[\lambda]) \in \{-1,1\}^\lambda$. Hence, for all $i \in [\lambda]$, it verifies whether $\widehat{B}_{j,k}[i] = g_{j,k}^{\hat{\alpha}_{j,k}[i]} \cdot T_{1,j,k}^{\hat{u}[i]}$ and $\widehat{H}_{j,k}[i] = h_{j,k}^{\hat{\alpha}_{j,k}[i]} \cdot T_{2,j,k}^{\hat{u}[i]}$. If for any $i \in [\lambda]$, the above verification fails, party $\mathsf{P}_k$ aborts; otherwise $\mathsf{P}_k$ accepts the commitment.

**Theorem 2.** *The protocol $\widehat{\Pi}_{\mathsf{sCOM}}$ srUC-realizes the $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$ functionality in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model in the presence of up to $n-1$ static malicious corruptions.*

*Proof.* To simplify notation, let $\widehat{\Pi} := \widehat{\Pi}_{\mathsf{sCOM}}$ and $\widehat{\mathcal{F}} := \widehat{\mathcal{F}}_{\mathsf{sCOM}}$. Recall that by definition of subversion resilience, we need to show that $\widehat{\Pi}$ UC-realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model, and that $\mathcal{F}$ can be written as a well-formed sanitizing ideal functionality. Towards this, we first build a simulator (communicating with $\widehat{\mathcal{F}}$) that simulates an execution of $\widehat{\Pi}$ for the case where $n-1$ parties are malicious, and the remaining party has an honest core and a semi-honest firewall. Note that, strictly speaking, one should also prove security for the case where there are less than $n-1$ malicious corruptions. It is, however, easy to see that proving the case with maximal corruption is complete. When the committer is honest then $\widehat{\mathcal{F}}$ gives the simulator the same power no matter how many receivers are corrupted, so assuming maximal corruption gives the adversary more power (without giving the simulator more power). When the committer is malicious, we need to simulate the view of at least one honest verifier (with a semi-honest core). Since the verifiers all act independently, it suffices to consider the case of maximal corruption.

**Lemma 3.** *For every malicious adversary $\mathcal{A}$ corrupting $n-1$ parties maliciously and the firewall of the remaining honest party semi-honestly in an execution of the protocol $\widehat{\Pi}$ in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{E}$:*

$$\textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}}_{\widehat{\Pi},\mathcal{A},\mathcal{E}} \approx \textsc{Exec}_{\widehat{\mathcal{F}},\mathcal{S},\mathcal{E}}.$$

*Proof.* We consider two cases, depending on the core of the committer being corrupt or not.

**Case 1: Malicious committer.** This corresponds to the case where the honest core $\mathsf{C}_k$ is the core of an honest verifier in an execution of the protocol $\widehat{\Pi}$. Denote with $\mathsf{C}_j$ and $\mathsf{F}_j$ the core and firewall corresponding to the maliciously corrupt committer. Here, the simulator $\mathcal{S}$ proceeds as follows.

**Setup:** Set up the CRS $\mathsf{crs}$ as in the real-world protocol. Additionally, the simulator $\mathcal{S}$ knows the trapdoor $t_{j,k}$ such that $h_{j,k} = g_{j,k}^{t_{j,k}}$.

**Commit phase:** Upon receiving a commitment $\hat{c}_{j,k} = (\hat{c}_{j,k}[1], \cdots, \hat{c}_{j,k}[\lambda])$ from $\mathcal{A}$, parse $\hat{c}_{j,k}[i] = (\widehat{B}_{j,k}[i], \widehat{H}_{j,k}[i])$. Thus, for all $i \in [\lambda]$, do:

- If $\widehat{H}_{j,k}[i] \cdot T_{2,j,k} = (\widehat{B}_{j,k}[i] \cdot T_{1,j,k})^{t_{j,k}}$, set $\hat{u}[i] = -1$.
- Else if $\widehat{H}_{j,k}[i]/T_{2,j,k} = (\widehat{B}_{j,k}[i]/T_{1,j,k})^{t_{j,k}}$, set $\hat{u}[i] = 1$.

Decode $\hat{u} \in \{-1, 1\}^\lambda$ into $\hat{s} \in \{0, 1\}^\lambda$. Finally, send $(\textsc{Commit}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_j, \hat{s})$ to $\widehat{\mathcal{F}}$ on behalf of $\mathsf{C}_j$ and send $(\textsc{Blind}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_j, 0^\lambda)$ to $\widehat{\mathcal{F}}$ on behalf of $\mathsf{F}_j$.

**Verification phase:** Upon receiving an opening $(s', \alpha'_{j,k})$ from $\mathcal{A}$, verify the opening as an honest verifier would do. If the verification succeeds send $(\textsc{Open}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_j)$ to $\widehat{\mathcal{F}}$; else, simulate $\mathcal{A}$ aborting and terminate.

We claim that the above simulation is perfect. In fact, the CRS $\mathsf{crs}$ is distributed exactly like in the real-world protocol. Moreover, since the tuple $\mathsf{crs}_{j,k} = (g_{j,k}, h_{j,k}, T_{1,j,k}, T_{2,j,k})$ is a non-DH tuple, the extraction procedure run by $\mathcal{S}$ always succeeds, which yields a perfect simulation of the commit phase. Finally, the verification phase is run exactly like in the real-world protocol.

**Case 2: Honest committer.** This corresponds to the case where $\mathsf{C}_j$ is the core of the honest committer, so that $\mathsf{F}_j$ is semi-honestly corrupt. Here, the simulator $\mathcal{S}$ proceeds as follows.

**Setup:** Set the CRS $\mathsf{crs}$ in such a way that the tuples $\mathsf{crs}_{j,k} = (g_{j,k}, h_{j,k}, T_{1,j,k}, T_{2,j,k})$ is a DH tuple. Namely, the simulator knows the trapdoor $x_{j,k}$ such that $T_{1,j,k} = g_{j,k}^{x_{j,k}}$ and $T_{2,j,k} = h_{j,k}^{x_{j,k}}$.

**Commit phase:** Sample a random $\alpha_{j,k} = (\alpha_{j,k}[1], \cdots, \alpha_{j,k}[\lambda]) \in \mathbb{Z}_q^\lambda$, and, for all $i \in [\lambda]$, let $B_{j,k}[i] = g_{j,k}^{\alpha_{j,k}[i]}$, $H_{j,k}[i] = h_{j,k}^{\alpha_{j,k}[i]}$ and $c_{j,k}[i] = (B_{j,k}[i], H_{j,k}[i])$. Furthermore, sample a random $\beta_{j,k} = (\beta_{j,k}[1], \cdots, \beta_{j,k}[\lambda]) \in \mathbb{Z}_q^\lambda$ and sanitize each $c_{j,k}[i]$ to get $\hat{c}_{j,k}[i] = (\widehat{B}_{j,k}[i], \widehat{H}_{j,k}[i])$ computed as follows:[3] If $r[i] = 0$, let $\widehat{B}_{j,k}[i] = B_{j,k}[i] \cdot g_{j,k}^{\beta_{j,k}[i]} = g_{j,k}^{\alpha_{j,k}[i]+\beta_{j,k}[i]}$ and $\widehat{H}_{j,k}[i] = H_{j,k}[i] \cdot h_{j,k}^{\beta_{j,k}[i]} = h_{j,k}^{\alpha_{j,k}[i]+\beta_{j,k}[i]}$. Else if $r[i] = 1$, let $\widehat{B}_{j,k}[i] = B_{j,k}[i]^{-1} \cdot g_{j,k}^{\beta_{j,k}[i]} = g_{j,k}^{-\alpha_{j,k}[i]+\beta_{j,k}[i]}$ and $\widehat{H}_{j,k}[i] = H_{j,k}[i]^{-1} \cdot h_{j,k}^{\beta_{j,k}[i]} = h_{j,k}^{-\alpha_{j,k}[i]+\beta_{j,k}[i]}$. Finally, send $\hat{c}_{j,k} = (\hat{c}_{j,k}[1], \cdots, \hat{c}_{j,k}[\lambda])$ to $\mathcal{A}$.

**Opening phase:** Upon receiving $(\textsc{Open}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_j, \hat{s})$, where $\hat{s} = (\hat{s}[1], \cdots, \hat{s}[\lambda]) \in \{0, 1\}^\lambda$, from $\widehat{\mathcal{F}}$, the simulator computes the corresponding encoding $\hat{u} = (\hat{u}[1], \cdots, \hat{u}[\lambda]) \in \{-1, 1\}^\lambda$ and adjusts the randomness by letting $\hat{\alpha}'_{j,k}[i] = \hat{\alpha}_{j,k}[i] - \hat{u}[i] \cdot x_{j,k}$ for all $i \in [\lambda]$. Here, $\hat{\alpha}_{j,k}[i] = \alpha_{j,k}[i] + \beta_{j,k}[i]$ if $r[i] = 0$, whereas $\hat{\alpha}_{j,k}[i] = -\alpha_{j,k}[i] + \beta_{j,k}[i]$ if $r[i] = 1$. Thus, the simulator sends $(\hat{s}, \hat{\alpha}'_{j,k})$ to $\mathcal{A}$, where $\hat{\alpha}'_{j,k} = (\hat{\alpha}'_{j,k}[1], \cdots, \hat{\alpha}'_{j,k}[\lambda])$.

[3] Note that the following steps can indeed be performed by the simulator since the firewall of $\mathsf{F}_j$ is semi-honestly corrupt, and thus $\mathcal{S}$ receives the blinding factor $r$ from the ideal functionality, as inputs of semi-honest parties are revealed.

There are two main differences between the above simulation and a real-world execution: (i) in the real world, the CRS $\mathsf{crs}$ is distributed like a sequence of non-DH tuples, whereas in the ideal world the simulator samples each $\mathsf{crs}_{j,k}$ as being a DH-tuple; (ii) in the real world, the sanitized commitment $\hat{c}_{j,k}$ is such that $(\widehat{B}_{j,k}[i], \widehat{H}_{j,k}[i])$ is a non-DH tuple for all $i \in [\lambda]$, whereas in the ideal world the simulated pair $(\widehat{B}_{j,k}[i], \widehat{H}_{j,k}[i])$ is a DH-tuple for all $i \in [\lambda]$. Note that nevertheless, the simulator can always adjust the randomness to match the string $\hat{s}$ used by $\mathsf{C}_j$ in the ideal world using the trapdoor $x_{j,k}$ (as described above). Hence, indistinguishability of the simulation follows by a standard hybrid argument using the DDH assumption. This finishes the proof of the lemma. □

Next, we show that the firewall $\mathsf{F}_j$ of the committer is strongly sanitizing (see Definition 10), meaning that a specious core behind the firewall looks like an honest core.

**Lemma 4.** *The firewall $\mathsf{F}_j$ of the committer $\mathsf{C}_j$ in $\widehat{\Pi}$ is strongly sanitizing.*

*Proof.* We need to show that for all poly-time environments $\mathcal{E}$ which do not corrupt the firewall $\mathsf{F}_j$, but which are allowed a specious corruption of the core $\mathsf{C}_j$, it holds that

$$\textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}}_{\widehat{\Pi},\mathcal{A},\mathcal{E}} \approx \textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}}_{\widehat{\Pi}',\mathcal{A},\mathcal{E}},$$

where $\mathcal{A}$ is the dummy adversary and where $\widehat{\Pi}$ and $\widehat{\Pi}'$ run with dummy parties except for $\mathsf{P}_j$ that is either taken to be $(\mathsf{C}_j, \mathsf{F}_j)$ or $(\widehat{\mathsf{C}}_j, \mathsf{F}_j)$ for an incorruptible core $\widehat{\mathsf{C}}_j$. Recall that when an incorruptible core receives a specious corruption $(\textsc{Specious}, \widetilde{\mathsf{C}}_j)$ from the environment, it ignores it and keeps running the code of $\mathsf{C}_j$.

Note that, in a real execution, the honest core $\mathsf{C}_j$ samples each value $\alpha_{j,k}[i]$ uniformly at random from $\mathbb{Z}_q$, and hence each $c_{j,k}[i] = (B_{j,k}[i], H_{j,k}[i])$ is uniformly random in $\mathbb{G}^2$, and so is the sanitized commitment $\hat{c}_{j,k}[i] = (\widehat{B}_{j,k}[i], \widehat{H}_{j,k}[i]])$. Moreover, we claim that for any commitment $\tilde{c}_{j,k} = (\tilde{c}_{j,k}[1], \cdots, \tilde{c}_{j,k}[\lambda])$ output by a specious core $\mathsf{C}_j$, except with negligible probability, there must exist values $\tilde{u}[i] \in \{-1, 1\}$ (and thus bits $\tilde{s}[i] \in \{0, 1\}$) such that $\tilde{c}_{j,k}$ can be opened to $\tilde{s} = (\tilde{s}[1], \cdots, \tilde{s}[\lambda])$. This is because otherwise, we can build a poly-time test $\mathsf{T}$ that tells apart non-rewinding black-box access to either $\widetilde{\mathsf{C}}_j$ or $\mathsf{C}_j$ by asking it to first compute and then open a commitment. This shows that a specious core, except with negligible probability, still outputs a well-formed commitment $\tilde{c}_{j,k}$; given such a commitment, the firewall $\mathsf{F}_j$ produces a sanitized committed that is uniformly random in $\mathbb{G}^2$. The lemma follows. □

The theorem statement now follows by looking at the standard corruption transition table used by the well-formed sanitizing ideal functionality $\widehat{\mathcal{F}}$. Since the adversary maliciously corrupts up to $n-1$ verifiers, there is at least one party which is the committer for which either (i) the core is honest and the firewall is semi-honest, or (ii) the core is specious and the firewall is honest. By Lemma 2, since an honest firewall is strongly sanitizing (as shown in Lemma 4), the core in case (ii) can be taken to be honest. Hence, the statement follows directly by Lemma 3. Note that here we are assuming that $\widehat{\mathcal{F}}$ treats a corruption with flavor $\textsc{Isolate}$ as a $\textsc{Malicious}$ corruption. □

# 4 Coin Tossing

In this section, we build a sanitizing protocol that implements the regular coin tossing functionality. Our protocol is described in the $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$-hybrid model, and therefore must implement the firewall that interacts with the $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$ functionality.

### 4.1 The Coin Tossing Functionality

We start by recalling the regular $\mathcal{F}_{\mathsf{TOSS}}$ functionality below. Intuitively, the functionality waits to receive an initialization message from all the parties. Hence, it samples a uniformly random $\lambda$-bit string $s$ and sends $s$ to the adversary. The adversary now can decide to deliver $s$ to a subset of the parties. The latter restriction comes from the fact that it is impossible to toss a coin fairly so that no adversary can cause a premature abort, or bias the outcome, without assuming honest majority [Cle86].

**Functionality $\mathcal{F}_{\mathsf{TOSS}}$**

The coin tossing functionality $\mathcal{F}_{\mathsf{TOSS}}$ runs with parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, and an adversary $\mathcal{S}$. It consists of the following communication interface.

- Upon receiving a message $(\textsc{Init}, \mathsf{sid}, \mathsf{P}_i)$ from $\mathsf{P}_i$: If this is the first such message from $\mathsf{P}_i$ then record $(\mathsf{sid}, \mathsf{P}_i)$ and send $(\textsc{Init}, \mathsf{P}_i)$ to $\mathcal{S}$. If there exist records $(\mathsf{sid}, \mathsf{P}_j)$ for all $(\mathsf{P}_j)_{j\in[n]}$, then sample a uniformly random bit string $s \in \{0,1\}^\lambda$ and send $s$ to the adversary $\mathcal{S}$.
- Upon receiving a message $(\textsc{Deliver}, \mathsf{sid}, \mathsf{P}_i)$ from $\mathcal{S}$ (and if this is the first such message from $\mathcal{S}$), and if there exist records $(\mathsf{sid}, \mathsf{P}_j)$ for all $(\mathsf{P}_j)_{j\in[n]}$, send $s$ to $\mathsf{P}_i$; otherwise do nothing.

### 4.2 Sanitizing Blum's Protocol

Next, we show how to sanitize a variation of the classical Blum coin tossing protocol [Blu81]. In this protocol, each party commits to a random string $s_i$ and later opens the commitment, thus yielding $s = s_1 \oplus \cdots \oplus s_n$. The firewall here samples an independent random string $r_i$ which is used to blind the string $s_i$ chosen by the (possibly specious) core.

**Protocol $\widehat{\Pi}_{\mathsf{TOSS}}$ (Sanitizing Blum's Coin Tossing)**

The protocol is described in the $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$-hybrid model, and is executed between parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ each consisting of a core $\mathsf{C}_i$ and a firewall $\mathsf{F}_i$. Party $\mathsf{P}_i = (\mathsf{C}_i, \mathsf{F}_i)$ proceeds as follows (the code for all other parties is analogous).

1. The core $\mathsf{C}_i$ samples a random string $s_i \in \{0,1\}^\lambda$ and sends $(\textsc{Commit}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, s_i)$ to $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$.
2. Upon receiving $(\textsc{Receipt}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i)$ from $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$, the firewall $\mathsf{F}_i$ samples a random string $r_i \in \{0,1\}^\lambda$ and sends $(\textsc{Blind}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, r_i)$ to $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$.
3. Upon receiving $(\textsc{Blinded}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, r_i)$ from $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$, as well as $(\textsc{Receipt}, \mathsf{sid}_j, \mathsf{cid}_j, \mathsf{C}_j)$ for all other cores $\mathsf{C}_{j\neq i}$, the core $\mathsf{C}_i$ sends the message $(\textsc{Open}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i)$ to $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$.
4. Upon receiving $(\textsc{Open}, \mathsf{sid}_j, \mathsf{cid}_j, \mathsf{C}_j, \hat{s}_j)$ from $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$, for each core $\mathsf{C}_{j\neq i}$, the core $\mathsf{C}_i$ outputs $s := s_i \oplus r_i \oplus \bigoplus_{j\neq i} \hat{s}_j$. (If any of the cores $\mathsf{C}_j$ do not open its commitment, then $\mathsf{C}_i$ sets $\hat{s}_j = 0^\lambda$.)

**Theorem 3.** *The protocol $\widehat{\Pi}_{\mathsf{TOSS}}$ wsrUC-realizes the $\mathcal{F}_{\mathsf{TOSS}}$ functionality in the $(\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{sCOM}})$-hybrid model in the presence of up to $n-1$ malicious corruptions.*

*Proof.* To simplify notation, let $\widehat{\Pi} := \widehat{\Pi}_{\mathsf{TOSS}}$ and $\mathcal{F} := \mathcal{F}_{\mathsf{TOSS}}$. Recall that by definition of wrapped subversion resilience, we need to show that $\widehat{\Pi}$ UC-realizes $\mathsf{Wrap}(\mathcal{F})$ in the $(\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{sCOM}})$-hybrid model. Towards this, we first build a simulator (communicating with $\mathsf{Wrap}(\mathcal{F})$) that simulates an execution of $\widehat{\Pi}$ for the case where $n-1$ parties are malicious, and the remaining party has an honest core and a semi-honest firewall. Note that, strictly speaking, one should also prove security for case where there are less than $n-1$ malicious corruptions. It is, however, easy to see that proving the case with maximal corruption is complete in the present case. The ideal functionality $\widehat{\mathcal{F}}$ gives the simulator the same powers no matter how many parties are corrupted, so assuming full corruption gives the adversary more powers (without giving the simulator more powers).

**Lemma 5.** *For every malicious adversary $\mathcal{A}$ corrupting $n-1$ parties maliciously and the firewall of the remaining honest party semi-honestly in an execution of $\widehat{\Pi}$ in the $(\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{sCOM}})$-hybrid model, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{E}$:*

$$\textit{Exec}^{\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{sCOM}}}_{\widehat{\Pi}, \mathcal{A}, \mathcal{E}} \equiv \textit{Exec}_{\mathsf{Wrap}(\mathcal{F}), \mathcal{S}, \mathcal{E}}.$$

*Proof.* Note that the coin tossing functionality has no inputs. Hence, the goal of the simulator is to make the output of the execution it simulates equal to the output that it receives from the coin tossing functionality. In what follows, we let $i \in [n]$ be the index corresponding to the only party with an honest core. We build the simulator $\mathcal{S}$ below.

**Commitment phase:** Upon receiving $(\textsc{Commit}, \mathsf{sid}_j, \mathsf{cid}_j, \mathsf{C}_j, s_j)$ from $\mathcal{A}$, for each $\mathsf{C}_{j \neq i}$, send $(\textsc{Receipt}, \mathsf{sid}_j, \mathsf{cid}_j, \mathsf{C}_j)$ to $\mathcal{A}$ and $(\textsc{Init}, \mathsf{sid}_j, \mathsf{P}_j)$ to $\mathcal{F}$. Upon receiving $(\textsc{Blind}, \mathsf{sid}_j, \mathsf{cid}_j, r_j)$ from $\mathcal{A}$, for each $\mathsf{C}_{j \neq i}$, send $(\textsc{Blinded}, \mathsf{sid}_j, \mathsf{cid}_j, \mathsf{C}_j, r_j)$ and $(\textsc{Receipt}, \mathsf{sid}_j, \mathsf{cid}_j, \mathsf{C}_j)$, to $\mathcal{A}$.

**Opening phase:** Upon receiving $s \in \{0,1\}^\lambda$ from $\mathcal{F}$, let $\hat{s}_i := \bigoplus_{j \neq i}(s_j \oplus r_j) \oplus r_i \oplus s$ and send $(\textsc{Open}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, \hat{s}_i)$ to $\mathcal{A}$. Upon receiving $(\textsc{Open}, \mathsf{sid}_j, \mathsf{cid}_j, \mathsf{C}_j)$ from $\mathsf{C}_j$, for each $\mathsf{C}_{j \neq i}$, send $(\textsc{Open}, \mathsf{sid}_j, \mathsf{cid}_j, \mathsf{C}_j, s_j \oplus r_j)$ to $\mathcal{A}$ and $(\textsc{Deliver}, \mathsf{sid}_j, \mathsf{P}_j)$ to $\mathcal{F}$.

We now argue that the simulation is perfect. The simulator $\mathcal{S}$ plays the role of the $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$ functionality, and hence it receives the inputs the malicious cores and firewalls send to $\widehat{\mathcal{F}}_{\mathsf{sCOM}}$. Furthermore, after receiving the coin tossing output $s$, the simulator can extract the sanitized input $\hat{s}_i$ of the honest core $\mathsf{C}_i$ by computing the xor between $s$, the sanitized strings $(s_j \oplus r_j)$ for each malicious core, and the blinding factor $r_i$ received from the semi-honest firewall $\mathsf{F}_i$. Let $\hat{s} := \bigoplus_{j \neq i}(s_j \oplus r_j) \oplus r_i$.

In a real execution, the honest core $\mathsf{C}_i$ would sample a uniformly random string $s_i \in \{0,1\}^\lambda$, independently of $\hat{s}$. In contrast, in an ideal execution, $s \in \{0,1\}^\lambda$ is chosen uniformly and then $s_i$ is set to be $\hat{s} \oplus s$. Since $s$ is chosen independently of $\hat{s}$, we have that $\hat{s} \oplus s$ is also uniformly distributed in $\{0,1\}^\lambda$. This concludes the proof. □

Next, we show that the firewall $\mathsf{F}_i$ of each party is strongly sanitizing (see Definition 10), meaning that a specious core behind the firewall looks like an honest core.

**Lemma 6.** *For each $i \in [n]$, the firewall $\mathsf{F}_i$ in $\widehat{\Pi}$ is strongly sanitising.*

*Proof.* We will show that for all environments $\mathcal{E}$ which do not corrupt the firewall $\mathsf{F}_i$, but which are allowed a specious corruption of the core $\mathsf{C}_i$, it holds that

$$\textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{sCOM}}}_{\widehat{\Pi}, \mathcal{A}, \mathcal{E}} \equiv \textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{sCOM}}}_{\widehat{\Pi}', \mathcal{A}, \mathcal{E}},$$

where $\mathcal{A}$ is the dummy adversary and where $\widehat{\Pi}$ and $\widehat{\Pi}'$ run with dummy parties except for $\mathsf{P}_i$ that is either taken to be $(\mathsf{C}_i, \mathsf{F}_i)$ or $(\widehat{\mathsf{C}}_i, \mathsf{F}_i)$ for an incorruptible core $\widehat{\mathsf{C}}_i$. Recall that when an incorruptible core receives a specious corruption $(\textsc{Specious}, \widetilde{\mathsf{C}}_i)$ from the environment, it ignores it and keeps running the code of $\mathsf{C}_i$.

Looking at $\widehat{\Pi}$, it is easy to see that $\hat{s}_i = s_i \oplus r_i$ is uniformly distributed in $\{0,1\}^\lambda$ as both $\mathsf{C}_i$ and $\mathsf{F}_i$ are honest. Now, a specious core $\widetilde{\mathsf{C}}_i$ can sample $s_i$ from a biased distribution. However, since $\mathsf{F}_i$ is honest, and it samples $r_i$ uniformly and independently of $s_i$, we have that $\hat{s}_i = s_i \oplus r_i$ is also uniformly distributed in $\{0,1\}^\lambda$. □

The theorem statement now follows by looking at the standard corruption transition table used by the functionality $\mathsf{Wrap}(\mathcal{F})$. Since the adversary maliciously corrupts up to $n-1$ parties, there is at least one party for which either (i) the core is honest and the firewall is semi-honest, or (ii) the core is specious and the firewalls is honest. By Lemma 2, since an honest firewall is strongly sanitizing (as shown in Lemma 6), the core in case (ii) can be taken to be honest. Hence, the statement follows directly by Lemma 5. Note that here we are assuming that $\mathsf{Wrap}(\mathcal{F})$ treats a corruption with flavor Isolate as a Malicious corruption; this is necessary, as if $\mathsf{P}_i$ is isolated and all other $\mathsf{P}_{j\neq i}$ are malicious the adversary can bias the output of the coin. □

# 5 Completeness Theorem

In this section, we show how to sanitize the classical compiler by Goldreich, Micali and Wigderson (GMW) [GMW87], for turning MPC protocols with security against *semi-honest* adversaries into ones with security against *malicious adversaries*. On a high level, the GMW compiler works by having each party commit to its input. Furthermore, the parties run a coin tossing protocol to determine the randomness to be used in the protocol; since the random tape of each party must be secret, the latter is done in such a way that the other parties only learn a commitment to the other parties' random tape. Finally, the commitments to each party's input and randomness are used to enforce semi-honest behavior: Each party computes the next message using the underlying semi-honest protocol, but also proves in zero knowledge that this message was computed correctly using the committed input and randomness.

## 5.1 Sanitizable Commit & Prove

The GMW compiler was analyzed in the UC setting by Canetti, Lindell, Ostrovsky and Sahai [CLOS02]. A difficulty that arises is that the receiver of a UC commitment obtains no information about the value that was committed to. Hence, the parties cannot prove in zero knowledge statements relative to their input/randomness commitment. This issue is solved by introducing a more general commit-and-prove functionality that essentially combines both the commitment and zero-knowledge capabilities in a single functionality. In turn, the commit-and-prove functionality can be realized using commitments and zero-knowledge proofs.

In order to sanitize the GMW compiler, we follow a similar approach. Namely, we introduce a sanitazable commit-and-prove functionality (denoted $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$ and depicted below) and show that this functionality suffices for our purpose. Intuitively, $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$ allows the core $\mathsf{C}_i$ of each party $\mathsf{P}_i$ to (i) commit to multiple secret inputs $x$, and (ii) prove arbitrary NP statements $y$ (w.r.t. an underlying relation $R$ that is a parameter of the functionality) whose corresponding witnesses consist of all the values $x$. Whenever the core $\mathsf{C}_i$ commits to a value $x$, the firewall $\mathsf{F}_i$ may decide to blind $x$ with a random string $r$ (which is then revealed to the core). Similarly, whenever the core proves a statement $y$, the firewall $\mathsf{F}_i$ may check if the given statement makes sense, in which case, and assuming the statement is valid, the functionality informs all other parties that $y$ is indeed a correct statement proven by $\mathsf{P}_i$.

**Functionality $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$**

The sanitizable commit-and-prove functionality $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$ is parameterized by an NP relation $R$, and runs with parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ (each consisting of a core $\mathsf{C}_i$ and a firewall $\mathsf{F}_i$) and an adversary $\mathcal{S}$. The functionality consists of the following communication interfaces for the cores and the firewalls respectively.

<u>**Interface IO**</u>

- Upon receiving a message $(\textsc{Commit}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, x)$ from $\mathsf{C}_i$, where $x \in \{0,1\}^*$, record the tuple

$(\mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, x)$ and send the message $(\textsc{Receipt}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i)$ to $\mathsf{F}_i$. Ignore future commands of the form $(\textsc{Commit}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \cdot)$.

- Upon receiving a message $(\textsc{Prove}, \mathsf{sid}, \mathsf{C}_i, y)$ from $\mathsf{C}_i$, if there is at least one record $(\mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \cdot)$ and a corresponding $(\textsc{Blind}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \cdot)$ message was sent to $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$, then send the message $(\textsc{Sanitize}, \mathsf{sid}, \mathsf{C}_i, y)$ to $\mathsf{F}_i$.

**Interface S**

- Upon receiving a message $(\textsc{Blind}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, r)$ from $\mathsf{F}_i$, where $r \in \{0,1\}^*$, proceed as follows: if the tuple $(\mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, x)$ is recorded, modify the tuple to be $(\mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \hat{x} = x \oplus r)$ and send the message $(\textsc{Blinded}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, r)$ to $\mathsf{C}_i$, and $(\textsc{Receipt}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i)$ to all $\mathsf{C}_{j \neq i}$ and $\mathcal{S}$; otherwise do nothing. Ignore future commands of the form $(\textsc{Blind}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_i, \cdot)$.
- Upon receiving a message $(\textsc{Continue}, \mathsf{sid}, \mathsf{C}_i, y)$ from $\mathsf{F}_i$, retrieve all tuples of the form $(\mathsf{sid}, \cdot, \mathsf{C}_i, \hat{x})$ and let $\overline{x}$ be the list containing all (possibly sanitized) witnesses $\hat{x}$. Then compute $R(y, \overline{x})$: if $R(y, \overline{x}) = 1$ send the message $(\textsc{Proved}, \mathsf{sid}, \mathsf{C}_i, y)$ to all $\mathsf{C}_{j \neq i}$ and $\mathcal{S}$, otherwise ignore the command.

In Appendix B, we show how to realize the sanitazable commit-and-prove functionality from *malleable dual-mode commitments*, a primitive which we introduce, and re-randomizable NIZKs for all of NP. Our commitment protocol from Section 3 can be seen as a concrete instantiation of malleable dual-mode commitments based on the DDH assumption.

## 5.2 Sanitizing the GMW Compiler

We are now ready to sanitize the GMW compiler. Let $\Pi$ be an MPC protocol. The (sanitized) protocol $\widehat{\Pi}_{\mathsf{GMW}}$ is depicted below and follows exactly the ideas outlined above adapted to the UC framework with reverse firewalls.

**Protocol $\widehat{\Pi}_{\mathsf{GMW}}$** (Sanitizing the GMW compiler)

The protocol is described in the $(\widehat{\mathcal{F}}_{\mathsf{C\&P}}, \mathcal{F}_{\mathsf{TOSS}})$-hybrid model, and is executed between parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ each consisting of a core $\mathsf{C}_i$ and a firewall $\mathsf{F}_i$. Party $\mathsf{P}_i = (\mathsf{C}_i, \mathsf{F}_i)$ proceeds as follows (the code for all other parties is analogous).

**Random tape generation:** When activated for the first time, party $\mathsf{P}_i$ generates its own randomness with the help of all other parties:

1. The core $\mathsf{C}_i$ picks a random $s_i \in \{0,1\}^\lambda$ and sends $(\textsc{Commit}, \mathsf{sid}_i, \mathsf{cid}_i, s_i)$ to $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$.
2. Upon receiving $(\textsc{Receipt}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i)$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$, the firewall $\mathsf{F}_i$ picks a random $r_i \in \{0,1\}^\lambda$ and sends $(\textsc{Blind}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, r_i)$ to $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$.
3. All the cores interact with $\mathcal{F}_{\mathsf{TOSS}}$ in order to obtain a public random string $s_i^*$ that is used to determine the random tape of $\mathsf{C}_i$. Namely, each core $\mathsf{C}_j$, for $j \in [n]$, sends $(\textsc{Init}, \mathsf{sid}_{i,j}, \mathsf{P}_j)$ to $\mathcal{F}_{\mathsf{TOSS}}$ and waits to receive the message $(\textsc{Delivered}, \mathsf{sid}_{i,j}, \mathsf{P}_j, s_i^*)$ from the functionality.
4. Upon receiving $(\textsc{Blinded}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, r_i)$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$, the core $\mathsf{C}_i$ defines $\hat{r}_i = s_i^* \oplus (s_i \oplus r_i)$.

**Input commitment:** When activated with input $x_i$, the core $\mathsf{C}_i$ sends $(\textsc{Commit}, \mathsf{sid}_i, \mathsf{cid}'_i, x_i)$ to $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$ and adds $x_i$ to the (initially empty) list of inputs $\overline{x}_i$ (containing the inputs from all the previous activations of the protocol). Upon receiving $(\textsc{Receipt}, \mathsf{sid}_i, \mathsf{cid}'_i, \mathsf{C}_i)$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$, the firewall $\mathsf{F}_i$ sends $(\textsc{Blind}, \mathsf{sid}_i, \mathsf{cid}'_i, \mathsf{C}_i, 0^{|x_i|})$ to $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$.

**Protocol execution:** Let $\tau \in \{0,1\}^*$ be the sequence of messages that were broadcast in all activations of $\Pi$ until now (where $\tau$ is initially empty).

1. The core $\mathsf{C}_i$ runs the code of $\Pi$ on its input list $\overline{x}_i$, transcript $\tau$, and random tape $\hat{r}_i$ (as determined above). If $\Pi$ instructs $\mathsf{P}_i$ to broadcast a message, proceed to the next step.
2. For each outgoing message $\mu_i$ that $\mathsf{P}_i$ sends in $\Pi$, the core $\mathsf{C}_i$ sends $(\textsc{Prove}, \mathsf{sid}_i, \mathsf{C}_i, (\mu_i, s_i^*, \tau))$ to $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$, where the relation parameterizing the functionality is defined as follows:

$$R := \{((\mu_i, s_i^*, \tau), (\overline{x}_i, s_i, r_i)) : \mu_i = \Pi(\overline{x}_i, \tau, s_i^* \oplus (s_i \oplus r_i))\}.$$

In words, the core $\mathsf{C}_i$ proves that the message $\mu_i$ is the correct next message generated by $\Pi$ when the input sequence is $\overline{x}_i$, the random tape is $\hat{r}_i = s_i^* \oplus (s_i \oplus r_i)$, and the current transcript is $\tau$. Thus, $\mathsf{C}_i$ appends $\mu_i$ to the current transcript $\tau$.

3. Upon receiving $(\textsc{Sanitize}, \mathsf{sid}_i, \mathsf{C}_i, (\mu_i, s_i^*, \tau))$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$, the firewall $\mathsf{F}_i$ verifies that $s_i^*$ is the same string obtained via $\mathcal{F}_{\mathsf{TOSS}}$ and that $\tau$ consists of all the messages that were broadcast in all the activations up to this point. If these conditions are not met, $\mathsf{F}_i$ ignores the message and otherwise it sends $(\textsc{Continue}, \mathsf{sid}_i, \mathsf{C}_i, (\mu_i, s_i^*, \tau))$ to $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$ and appends $\mu_i$ to the current transcript $\tau$.
4. Upon receiving $(\textsc{Proved}, \mathsf{sid}_j, \mathsf{C}_j, (\mu_j, s_i^*, \tau))$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$, both the core $\mathsf{C}_i$ and the firewall $\mathsf{F}_i$ append $\mu_j$ to the transcript $\tau$ and repeat the above steps.

**Output:** Whenever $\Pi$ outputs a value, $\widehat{\Pi}_{\mathsf{GMW}}$ generates the same output.

A few remarks are in order. First, and without loss of generality, we assume that the underlying protocol $\Pi$ is reactive and works by a series of activations, where in each activation, only one of the parties has an input; the random tape of each party is taken to be a $\lambda$-bit string for simplicity. Second, each party needs to invoke an independent copy of $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$; we identify these copies as $\mathsf{sid}_i$, where we can for instance let $\mathsf{sid}_i = \mathsf{sid}||i$. Third, we slightly simplify the randomness generation phase using the coin tossing functionality $\mathcal{F}_{\mathsf{TOSS}}$. In particular, each core $\mathsf{C}_i$ commits to a random string $s_i$ via $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$; the corresponding firewall $\mathsf{F}_i$ blinds $s_i$ with a random string $r_i$. Thus, the parties obtain public randomness $s_i^*$ via $\mathcal{F}_{\mathsf{TOSS}}$, yielding a sanitized random tape $\hat{r}_i = s_i^* \oplus (s_i \oplus r_i)$ for party $\mathsf{P}_i$. Note that it is crucial that the parties obtain independent public random strings $s_i^*$ in order to determine the random tape of party $\mathsf{P}_i$. In fact, if instead we would use a single invocation of $\mathcal{F}_{\mathsf{TOSS}}$ yielding common public randomness $s$, two malicious parties $\mathsf{P}_i$ and $\mathsf{P}_j$ could pick the same random tape by choosing the same values $s_i, r_i, s_j, r_j$. Clearly, the latter malicious adversary cannot be reduced to a semi-honest adversary.
The theorem below states the security of the GMW compiler with reverse firewalls.

**Theorem 4.** *Let $\mathcal{F}$ be any functionality for $n$ parties. Assuming that $\Pi$ UC realizes $\mathcal{F}$ in the presence of up to $t \leq n-1$ semi-honest corruptions, then the compiled protocol $\widehat{\Pi}_{\mathsf{GMW}}$ wsrUC realizes $\mathcal{F}$ in the $(\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{C\&P}}, \mathcal{F}_{\mathsf{TOSS}})$-hybrid model in the presence of up to $t$ malicious corruptions.*

*Proof.* Recall that, by definition of wrapped subversion resilience, we need to show that $\widehat{\Pi}_{\mathsf{GMW}}$ UC realizes $\mathsf{Wrap}(\mathcal{F})$ in the $(\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{C\&P}}, \mathcal{F}_{\mathsf{TOSS}})$-hybrid model. Towards this, we first prove that every adversary attacking $\widehat{\Pi}_{\mathsf{GMW}}$ in the $(\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{C\&P}}, \mathcal{F}_{\mathsf{TOSS}})$-hybrid model by corrupting up to $t$ parties maliciously, and the firewall of the remaining $n-t$ parties semi-honestly, can be simulated by an adversary attacking $\Pi$ by corrupting $t$ semi-honest parties.

**Lemma 7.** *For every adversary $\mathcal{B}$ that corrupts up to $t$ parties maliciously and the firewall of the remaining honest parties semi-honestly in an execution of $\widehat{\Pi}_{\mathsf{GMW}}$ in the $(\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{C\&P}}, \mathcal{F}_{\mathsf{TOSS}})$-hybrid model, there exists an adversary $\mathcal{A}$ that corrupts up to $t$ parties semi-honestly in an execution of $\Pi$, such that for all environments $\mathcal{E}$:*

$$\textsc{Exec}_{\Pi,\mathcal{A},\mathcal{E}} \equiv \textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}, \widehat{\mathcal{F}}_{\mathsf{C\&P}}, \mathcal{F}_{\mathsf{TOSS}}}_{\widehat{\Pi}_{\mathsf{GMW}},\mathcal{B},\mathcal{E}}.$$

*Proof.* We construct a semi-honest adversary $\mathcal{A}$ from the malicious adversary $\mathcal{B}$ that also corrupts semi-honestly the firewalls of the honest cores. The adversary $\mathcal{A}$ runs $\Pi$ while internally simulating an execution of $\widehat{\Pi}_{\mathsf{GMW}}$ for $\mathcal{B}$ in the hybrid model. In particular, the adversary $\mathcal{A}$ runs $\mathcal{B}$ and proceeds as follows.

**Communication with the environment:** The input values received by $\mathcal{A}$ from $\mathcal{E}$ are written on $\mathcal{B}$'s input tape, and the output values of $\mathcal{B}$ are copied to $\mathcal{A}$'s own output tape.

**Randomness generation phase:** When the first activation of $\Pi$ takes place, $\mathcal{A}$ simulates the random tape generation phase of $\widehat{\Pi}_{\mathsf{GMW}}$ for $\mathcal{B}$. In particular, the simulation below is repeated for every party $\mathsf{P}_i = (\mathsf{C}_i, \mathsf{F}_i)$.

- *Honest* $\mathsf{C}_i$ *and semi-honest* $\mathsf{F}_i$*:* The adversary $\mathcal{A}$ internally hands $\mathcal{B}$ the message $(\textsc{Receipt}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i)$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. Upon receiving $(\textsc{Blind}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, r_i)$ from $\mathsf{F}_i$, the adversary $\mathcal{A}$ internally hands $\mathcal{B}$ the message $(\textsc{Blinded}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, r_i)$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. In addition, $\mathcal{A}$ simulates all the $(\textsc{Receipt}, \mathsf{sid}_j, \mathsf{cid}_j, \mathsf{C}_j)$ messages that $\mathcal{B}$ expects to receive from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$.
- *Malicious* $\mathsf{P}_i$*:* Upon receiving $(\textsc{Commit}, \mathsf{sid}_i, \mathsf{cid}_i, s_i)$ from $\mathcal{B}$, the adversary $\mathcal{A}$ internally hands $\mathcal{B}$ the message $(\textsc{Receipt}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i)$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. Upon receiving $(\textsc{Blind}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, r_i)$ from $\mathcal{B}$, the adversary $\mathcal{A}$ internally hands $\mathcal{B}$ the message $(\textsc{Blinded}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, r_i)$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. In addition, $\mathcal{A}$ simulates all the $(\textsc{Receipt}, \mathsf{sid}_j, \mathsf{cid}_j, \mathsf{C}_j)$ messages that $\mathcal{B}$ expects to receive from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. Finally, upon receiving $(\textsc{Init}, \mathsf{sid}_{i,j}, \mathsf{P}_i)$ from $\mathcal{B}$, for all indexes $j \in [n]$ corresponding to a party $\mathsf{P}_j$ under control of $\mathcal{B}$, the adversary $\mathcal{A}$ hands $(\textsc{Deliver}, \mathsf{sid}_{i,j}, \mathsf{P}_i, s_i^*)$ to $\mathcal{B}$ (on behalf of $\mathcal{F}_{\mathsf{TOSS}}$), where $s_i^* \in \{0,1\}^\lambda$ is a random string.

**Input commitment:** When the first message of an activation of $\Pi$ is sent, $\mathcal{A}$ internally simulates for $\mathcal{B}$ the appropriate stage in $\widehat{\Pi}_{\mathsf{GMW}}$. This is done as follows. Let $\mathsf{P}_i$ be the activated party with a new input.

- *Honest* $\mathsf{C}_i$ *and semi-honest* $\mathsf{F}_i$*:* The adversary $\mathcal{A}$ internally hands $\mathcal{B}$ the message $(\textsc{Receipt}, \mathsf{sid}_i, \mathsf{cid}'_i, \mathsf{C}_i)$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. Upon receiving $(\textsc{Blind}, \mathsf{sid}_i, \mathsf{cid}'_i, \mathsf{C}_i, 0^{|x_i|})$ from $\mathsf{F}_i$, the adversary $\mathcal{A}$ internally hands $\mathcal{B}$ the message $(\textsc{Blinded}, \mathsf{sid}_i, \mathsf{cid}'_i, \mathsf{C}_i, 0^{|x_i|})$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. In addition, $\mathcal{A}$ simulates all the $(\textsc{Receipt}, \mathsf{sid}_j, \mathsf{cid}'_j, \mathsf{C}_j)$ messages that $\mathcal{B}$ expects to receive from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$.
- *Malicious* $\mathsf{P}_i$*:* Upon receiving $(\textsc{Commit}, \mathsf{sid}_i, \mathsf{cid}'_i, x_i)$ from $\mathcal{B}$, the adversary $\mathcal{A}$ internally hands $\mathcal{B}$ the message $(\textsc{Receipt}, \mathsf{sid}_i, \mathsf{cid}'_i, \mathsf{C}_i)$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. Upon receiving $(\textsc{Blind}, \mathsf{sid}_i, \mathsf{cid}'_i, \mathsf{C}_i, r'_i)$ from $\mathcal{B}$, the adversary $\mathcal{A}$ internally hands $\mathcal{B}$ the message $(\textsc{Blinded}, \mathsf{sid}_i, \mathsf{cid}'_i, \mathsf{C}_i, r'_i)$ from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. In addition, $\mathcal{A}$ simulates all the $(\textsc{Receipt}, \mathsf{sid}_j, \mathsf{cid}'_j, \mathsf{C}_j)$ messages that $\mathcal{B}$ expects to receive from $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$. Finally, $\mathcal{A}$ adds $x_i \oplus r'_i$ to its list $\overline{x}_i$ of inputs received from $\mathsf{P}_i$ and sets $\mathsf{P}_i$'s input tape to $x_i \oplus r'_i$.

**Protocol execution:** When an honest party $\mathsf{P}_i$ sends a message $\mu_i$ in $\Pi$ to a corrupted party (controlled by $\mathcal{A}$), then $\mathcal{A}$ prepares a simulated message for $\widehat{\Pi}_{\mathsf{GMW}}$ to give to $\mathcal{B}$. In particular, $\mathcal{A}$ passes $\mathcal{B}$ the message $(\textsc{Sanitize}, \mathsf{sid}_i, \mathsf{C}_i, (\mu_i, s_i^*, \tau))$ on behalf of $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$, where $\tau$ is the current transcript of the protocol. Upon receiving $(\textsc{Continue}, \mathsf{sid}_i, \mathsf{C}_i, (\mu_i, s_i^*, \tau))$ from $\mathcal{B}$, the adversary $\mathcal{A}$ sends $\mathcal{B}$ the message $(\textsc{Proved}, \mathsf{sid}_i, \mathsf{C}_i)$ on behalf of $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$.

When $\mathcal{B}$ sends a message $\mu_i$ from a malicious party, $\mathcal{A}$ translates this to the appropriate message in $\Pi$. In particular, $\mathcal{A}$ obtains a message $(\textsc{Prove}, \mathsf{sid}_i, \mathsf{C}_i, (\mu_i, s_i^*, \tau))$ from $\mathcal{B}$ on behalf of a corrupted party $\mathsf{P}_i$ to which $\mathcal{A}$ replies with $(\textsc{Sanitize}, \mathsf{sid}_i, \mathsf{C}_i, (\mu_i, s_i^*, \tau))$. Then, upon receiving $(\textsc{Continue}, \mathsf{sid}_i, \mathsf{C}_i, (\mu_i, s_i^*, \tau_i))$, adversary $\mathcal{A}$ checks that $\tau$ is indeed the current transcript, that $s_i^*$ is the random string sent earlier on behalf of $\mathcal{F}_{\mathsf{TOSS}}$, and that $R((\mu_i, s_i^*, \tau), (\overline{x}_i, s_i, r_i)) = 1$. Note that $\mathcal{A}$ can evaluate the relation $R$ as it received the values $\overline{x}_i, s_i, r_i$ from $\mathcal{B}$ (on behalf of malicious party $\mathsf{P}_i$). If all the checks pass, then $\mathcal{A}$ delivers $\mathcal{B}$ the message $(\textsc{Proved}, \mathsf{sid}_i, \mathsf{C}_i, (\mu_i, s_i^*, \tau))$ and finally writes $\mu_i$ on semi-honest party $\mathsf{P}_i$'s outgoing communication tape in $\Pi$. Otherwise, $\mathcal{A}$ does nothing.

We now claim that $\mathcal{E}$'s view in an interaction with $\mathcal{A}$ and $\Pi$ is distributed identically to its view in an interaction with $\mathcal{B}$ and $\widehat{\Pi}_{\mathsf{GMW}}$ in the $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$-hybrid model. The key points are as follows:

- Adversary $\mathcal{A}$ sets the randomness of each party $\mathsf{P}_i$ in an internal emulation of $\widehat{\Pi}_{\mathsf{GMW}}$ to $\hat{r}_i = s_i^* \oplus (s_i \oplus r_i)$. In case $\mathsf{C}_i$ is honest and $\mathsf{F}_i$ is semi-honest, the distribution of $\hat{r}_i$ is uniform from the point of view of $\mathcal{E}$ because $s_i$ is uniform and independent of both $r_i, s$ (which are known by $\mathcal{E}$). In case $\mathsf{P}_i$ is malicious, the distribution of $\hat{r}_i$ is uniform from the point of view of $\mathcal{E}$ because $s_i^*$ is uniform and independent of both $r_i, s_i$ (which are chosen by $\mathcal{E}$). Hence, $\mathcal{A}$ forces the randomness of each party $\mathsf{P}_i$ in an internal emulation of $\widehat{\Pi}_{\mathsf{GMW}}$ to be identically distributed to the randomness of a semi-honest party $\mathsf{P}_i$ in a run of $\Pi$.

- Adversary $\mathcal{A}$ modifies the input tape of each semi-honest party $\mathsf{P}_i$ to be the same input as committed to by $\mathcal{B}$. Note that this adjustment accounts for any non-zero blinding factor $r_i'$ that a malicious firewall may forward in $\widehat{\Pi}_{\mathsf{GMW}}$. As a consequence, the input and random tapes that the malicious $\mathcal{B}$ committed to on behalf of malicious $\mathsf{P}_i$ are exactly the same as the input and random tapes used by $\mathcal{A}$ on behalf of semi-honest $\mathsf{P}_i$.

- Adversary $\mathcal{A}$ is able to verify at every step if the message $\mu_i$ sent by $\mathcal{B}$, on behalf of malicious $\mathsf{P}_i$, is according to the protocol specification. If the check goes through, then it is guaranteed that $\mathsf{P}_i$ generates the exact same message $\mu_i$ in the external execution of $\Pi$. Thus, the other parties receive the same message in the execution of $\Pi$ (where the adversary $\mathcal{A}$ is semi-honest) and in the execution of $\widehat{\Pi}_{\mathsf{GMW}}$ (where the adversary $\mathcal{B}$ is malicious). Note that $\mathcal{A}$ does not need to make this check in case the core $\mathsf{C}_i$ is honest and the firewall $\mathsf{F}_i$ is semi-honest, as an honest core always proves a true statement and the semi-honest firewall sanitizes it (via $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$) without modifying the statement. Furthermore, it is guaranteed that whenever $\mathcal{A}$ delivers a message $\mu_i$ in the external execution of $\Pi$, the simulated $\mathcal{B}$ generated and delivered a valid corresponding message to $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$.

We conclude that the ensembles $\textsc{Exec}_{\Pi,\mathcal{A},\mathcal{E}}$ and $\textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}},\widehat{\mathcal{F}}_{\mathsf{C\&P}},\mathcal{F}_{\mathsf{TOSS}}}_{\widehat{\Pi}_{\mathsf{GMW}},\mathcal{B},\mathcal{E}}$ are identical. This completes the proof. □

Next, we show that the firewall $\mathsf{F}_i$ of each party is strongly sanitizing (see Definition 10), meaning that a specious core behind the firewall looks like an honest core.

**Lemma 8.** *For each $i \in [n]$, the firewall $\mathsf{F}_i$ in $\widehat{\Pi}_{\mathsf{GMW}}$ is strongly sanitizing.*

*Proof.* We need to show that for all poly-time environments $\mathcal{E}$ which do not corrupt the firewall $\mathsf{F}_i$, but which are allowed a specious corruption of the core $\mathsf{C}_i$, it holds that

$$\textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}},\widehat{\mathcal{F}}_{\mathsf{C\&P}},\mathcal{F}_{\mathsf{TOSS}}}_{\widehat{\Pi}_{\mathsf{GMW}},\mathcal{B},\mathcal{E}} \approx \textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}},\widehat{\mathcal{F}}_{\mathsf{C\&P}},\mathcal{F}_{\mathsf{TOSS}}}_{\widehat{\Pi}'_{\mathsf{GMW}},\mathcal{B},\mathcal{E}},$$

where $\mathcal{B}$ is the dummy adversary and where $\widehat{\Pi}_{\mathsf{GMW}}$ and $\widehat{\Pi}'_{\mathsf{GMW}}$ run with dummy parties except for $\mathsf{P}_i$ that is either taken to be $(\mathsf{C}_i, \mathsf{F}_i)$ or $(\widehat{\mathsf{C}}_i, \mathsf{F}_i)$ for an incorruptible core $\widehat{\mathsf{C}}_i$. Recall that when an incorruptible core receives a specious corruption $(\textsc{Specious}, \widetilde{\mathsf{C}}_i)$ from the environment, it ignores it and keeps running the code of $\mathsf{C}_i$.

The proof is by contradiction. Namely, assume that there exists a poly-time environment $\mathcal{E}$ that can tell apart the above two ensembles using a specious corruption $\widetilde{\mathsf{C}}_i$. We show how to build a poly-time test $\mathsf{T}$ that tells apart non-rewinding black-box access to either $\widetilde{\mathsf{C}}_i$ or $\mathsf{C}_i$. This contradicts the fact that $\widetilde{\mathsf{C}}_i$ is specious. The test $\mathsf{T}$ simply uses its target oracle to emulate a run of the protocol with dummy parties and honest firewall $\mathsf{F}_i$; in particular, since $\mathsf{F}_i$ is honest, the values $s_i$ received from the target oracle as part of $(\textsc{Commit}, \mathsf{sid}_i, \mathsf{cid}_i, \mathsf{C}_i, s_i)$ messages are always blinded with uniformly random values $r_i$, yielding a uniform random tape $\hat{r}_i$. This yields a transcript that is either distributed according to $\widehat{\Pi}_{\mathsf{GMW}}$ or to $\widehat{\Pi}'_{\mathsf{GMW}}$ depending on the target being $\widetilde{\mathsf{C}}_i$ or $\mathsf{C}_i$. Hence,$\mathsf{T}$ runs $\mathcal{E}$ on the simulated transcript and outputs whatever $\mathcal{E}$ outputs. This finishes the proof. □

The theorem statement now follows by looking at the standard corruption transition table used by the functionality $\mathsf{Wrap}(\mathcal{F})$. Since the adversary maliciously corrupts up to $t$ parties, there are at most $n - t$ parties for which either (i) the core is honest and the firewall is semi-honest, or (ii) the core is specious and the firewalls is honest. By Lemma 2, since honest firewalls are strongly sanitizing (as shown in Lemma 8), the cores in case (ii) can be taken to be honest. Hence, the statement follows directly by Lemma 7. Note that here we are assuming that $\mathsf{Wrap}(\mathcal{F})$ treats a corruption with flavor ISOLATE as a MALICIOUS corruption; this is necessary, as there are examples of protocols $\Pi$ and functionalities $\mathcal{F}$ for which $\widehat{\Pi}_{\mathsf{GMW}}$ simply becomes insecure if $t$ parties are malicious and $n - t$ parties are isolated (the Blum's protocol with the $\mathcal{F}_{\mathsf{TOSS}}$ functionality from Section 4 is such an example). □

## 6 Conclusions and Future Work

We have put forward a generalization of the UC framework by Canetti [Can01, Can00], where each party consists of a core (which has secret inputs and is in charge of generating protocol messages) and a reverse firewall (which has no secrets and sanitizes the outgoing/incoming communication from/to the core). Both the core and the firewall can be subject to different flavors of corruption, modeling the strongly adversarial setting where a subset of the players is maliciously corrupt, whereas the remaining honest parties are subject to subversion attacks. The main advantage of our approach is that it comes with very strong composition guarantees, as it allows, for the first time, to design subversion-resilient protocols that can be used as part of larger, more complex protocols, while retaining security even when protocol sessions are running concurrently (under adversarial scheduling) and in the presence of subversion attacks.

Moreover, we have demonstrated the feasibility of our approach by designing UC reverse firewalls for cryptographic protocols realizing pretty natural ideal functionalities such as commitments and coin tossing, and, in fact, even for arbitrary functionalities. Several avenues for further research are possible, including designing UC reverse firewalls for other ideal functionalities (such as oblivious transfer and zero knowledge), removing (at least partially) trusted setup assumptions, and defining UC subversion-resilient MPC in the presence of adaptive corruptions.

## References

[ABLZ17] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2017.

[AFMV19] Giuseppe Ateniese, Danilo Francati, Bernardo Magri, and Daniele Venturi. Public immunization against complete subversion without random oracles. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 465–485. Springer, Heidelberg, June 2019.

[ALSZ20] Behzad Abdolmaleki, Helger Lipmaa, Janno Siim, and Michal Zajac. On QA-NIZK in the BPK model. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 590–620. Springer, Heidelberg, May 2020.

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 364–375. ACM Press, October 2015.

[AsV08] Joël Alwen, abhi shelat, and Ivan Visconti. Collusion-free protocols in the mediated model. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 497–514. Springer, Heidelberg, August 2008.

[BBF$^+$20] Angèle Bossuat, Xavier Bultel, Pierre-Alain Fouque, Cristina Onete, and Thyla van der Merwe. Designing reverse firewalls for the real world. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part I*, volume 12308 of *LNCS*, pages 193–213. Springer, Heidelberg, September 2020.

[BCJ21] Pascal Bemmann, Rongmao Chen, and Tibor Jager. Subversion-resilient public key encryption with practical watchdogs. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 627–658. Springer, Heidelberg, May 2021.

[BDI$^+$99] Mike Burmester, Yvo Desmedt, Toshiya Itoh, Kouichi Sakurai, and Hiroki Shizuya. Divertible and subliminal-free zero-knowledge proofs for languages. *Journal of Cryptology*, 12(3):197–223, June 1999.

[BFS16] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, December 2016.

[BJK15] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1431–1440. ACM Press, October 2015.

[BL17] Sebastian Berndt and Maciej Liskiewicz. Algorithm substitution attacks from a steganographic perspective. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1649–1660. ACM Press, October / November 2017.

[Blu81] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, *CRYPTO'81*, volume ECE Report 82-04, pages 11–15. U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981.

[BPR14] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 1–19. Springer, Heidelberg, August 2014.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CDN20] Suvradip Chakraborty, Stefan Dziembowski, and Jesper Buus Nielsen. Reverse firewalls for actively secure MPCs. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 732–762. Springer, Heidelberg, August 2020.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

[CGPS21] Suvradip Chakraborty, Chaya Ganesh, Mahak Pancholi, and Pratik Sarkar. Reverse firewalls for adaptively secure mpc without setup. To Appear in *ASIACRYPT 2021*, 2021. https://ia.cr/2021/1262.

[CHY20] Rongmao Chen, Xinyi Huang, and Moti Yung. Subvert KEM to break DEM: Practical algorithm-substitution attacks on public-key encryption. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 98–128. Springer, Heidelberg, December 2020.

[CKLM12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 281–300. Springer, Heidelberg, April 2012.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[CMY+16] Rongmao Chen, Yi Mu, Guomin Yang, Willy Susilo, Fuchun Guo, and Mingwu Zhang. Cryptographic reverse firewall via malleable smooth projective hash functions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 844–876. Springer, Heidelberg, December 2016.

[CRT+19] Sherman S. M. Chow, Alexander Russell, Qiang Tang, Moti Yung, Yongjun Zhao, and Hong-Sheng Zhou. Let a non-barking watchdog bite: Cliptographic signatures with an offline watchdog. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part I*, volume 11442 of *LNCS*, pages 221–251. Springer, Heidelberg, April 2019.

[CSW20] Ran Canetti, Pratik Sarkar, and Xiao Wang. Efficient and round-optimal oblivious transfer and commitment with adaptive security. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 277–308. Springer, Heidelberg, December 2020.

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

[DGG+15] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 101–126. Springer, Heidelberg, April 2015.

[DMS16] Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. Message transmission with reverse firewalls—secure communication on corrupted machines. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 341–372. Springer, Heidelberg, August 2016.

[DPSW16] Jean Paul Degabriele, Kenneth G. Paterson, Jacob C. N. Schuldt, and Joanne Woodage. Backdoors in pseudorandom number generators: Possibility and impossibility results. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 403–432. Springer, Heidelberg, August 2016.

[FJM18] Marc Fischlin, Christian Janson, and Sogol Mazaheri. Backdoored hash functions: Immunizing HMAC and HKDF. In Steve Chong and Stephanie Delaune, editors, *CSF 2018 Computer Security Foundations Symposium*, pages 105–118. IEEE Computer Society Press, 2018.

[FM18] Marc Fischlin and Sogol Mazaheri. Self-guarding cryptographic protocols against algorithm substitution attacks. In Steve Chong and Stephanie Delaune, editors, *CSF 2018 Computer Security Foundations Symposium*, pages 76–90. IEEE Computer Society Press, 2018.

[Fuc18] Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg, March 2018.

[GMV20] Chaya Ganesh, Bernardo Magri, and Daniele Venturi. Cryptographic reverse firewalls for interactive proof systems. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *ICALP 2020*, volume 168 of *LIPIcs*, pages 55:1–55:16. Schloss Dagstuhl, July 2020.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[LMs05] Matt Lepinski, Silvio Micali, and abhi shelat. Collusion-free protocols. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 543–552. ACM Press, May 2005.

[MS15] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 657–686. Springer, Heidelberg, April 2015.

[OO90] Tatsuaki Okamoto and Kazuo Ohta. Divertible zero knowledge interactive proofs and commutative random self-reducibility. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 134–148. Springer, Heidelberg, April 1990.

[RTYZ16] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 34–64. Springer, Heidelberg, December 2016.

[RTYZ17] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 907–922. ACM Press, October / November 2017.

[RTYZ18] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Correcting subverted random oracles. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 241–271. Springer, Heidelberg, August 2018.

[Sim84] Gustavus J. Simmons. Authentication theory/coding theory. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 411–431. Springer, Heidelberg, August 1984.

[Sim86] Gustavus J. Simmons. A secure subliminal channel (?). In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 33–41. Springer, Heidelberg, August 1986.

[YY96] Adam Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 89–103. Springer, Heidelberg, August 1996.

[YY97] Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 62–74. Springer, Heidelberg, May 1997.

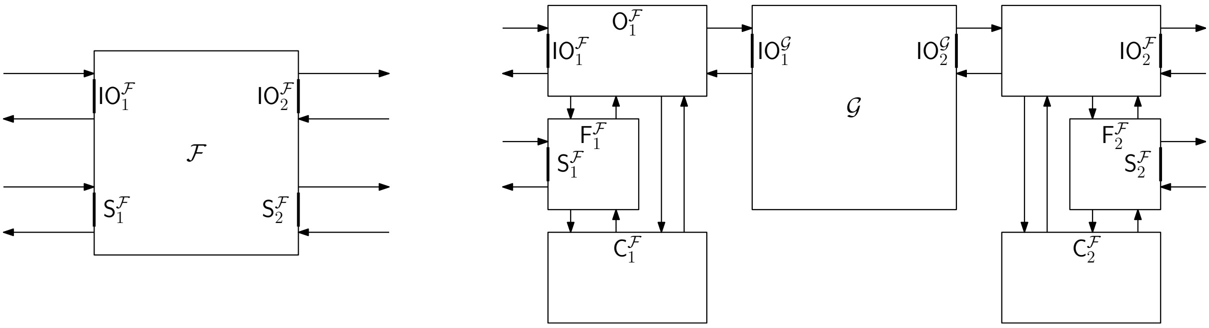

$\mathsf{IO}_1^{\mathcal{F}}$
$\mathsf{IO}_2^{\mathcal{F}}$
$\mathcal{F}$
$\mathsf{S}_1^{\mathcal{F}}$
$\mathsf{S}_2^{\mathcal{F}}$
$\mathsf{O}_1^{\mathcal{F}}$
$\mathsf{IO}_1^{\mathcal{F}}$
$\mathsf{IO}_1^{\mathcal{G}}$
$\mathsf{IO}_2^{\mathcal{G}}$
$\mathcal{G}$
$\mathsf{IO}_2^{\mathcal{F}}$
$\mathsf{F}_1^{\mathcal{F}}$
$\mathsf{S}_1^{\mathcal{F}}$
$\mathsf{F}_2^{\mathcal{F}}$
$\mathsf{S}_2^{\mathcal{F}}$
$\mathsf{C}_1^{\mathcal{F}}$
$\mathsf{C}_2^{\mathcal{F}}$

Figure 8: Illustration of a three tier model.

# A The Three-Tier Model

In this work we have assumed sanitisable authenticated transfer $\mathcal{F}_{\mathsf{SAT}}$ as the underlying communication network. One can further imagine implementing $\mathcal{F}_{\mathsf{SAT}}$ on top of unauthenticated transfer and a firewall sanitizing the communication. In this case, our simple two-tier model with just a core and a firewall would however come to its limit. The purpose of this section is to discuss this limit, sketch a way to mitigate it, and discuss why we have anyway chosen to study the two-tier model.

We have chosen the two-tier model as our main model of study as it is the minimal model having both interesting aspects of subversion resilience, namely a core with secrets and a sanitizing firewall without secrets. However, in the case of authenticated transfer as the communication network we get the problem that the firewall would isolate the core from the unauthenticated network. To ensure communication is not manipulated in transfer between parties we need to authenticate. We cannot let the firewall authenticate alone as it should not keep secret keys. We cannot let the core do the authentication as we need that the firewall can change the messages. It therefore becomes more natural to assume a three-tier model where there is also an *operative* component $\mathsf{O}_i$ coordinating communication between core, firewall and network. It is then natural to say that the operative handles inputs and outputs and communication with the network. The firewall would then sit between the core and the operative. The operative could also need to have direct access to the core. This is illustrated in Fig. 8.

The intended use would be to let the core keep secret keys of cryptographic algorithms and offer an API to use them. The firewall would sanitize outgoing messages to protect the keys. The operative would handle the "non-cryptographic" part of the protocol. One could again study different combinations of corruptions. It is natural to assume that the operative is honest or malicious, that the core is honest, specious or malicious, and that the firewall is honest, semi-honest or malicious.

We note that, in the three-tier model, we could implement $\mathcal{F}_{\mathsf{SAT}}$ using subversion-resilient signatures as follows.

- Assume that $\mathcal{G}$ models a PKI plus unauthenticated communication, *i.e.*, it offers a PKI which initially allows all parties to broadcast a public key and after that it allows unauthenticated transfer where the adversary can manipulate the communication.

- Assume that the core is a box which given a secret key $\mathsf{sk}$ will produce signatures $\sigma$ under this secret key.

- The operative will initially generate a key pair $(\mathsf{pk}, \mathsf{sk})$ for a signature scheme, put $\mathsf{sk}$ inside the core and broadcast $\mathsf{pk}$ using the PKI.

- Upon input a message $\mu$ on $\mathsf{IO}_i^{\mathcal{F}_{\mathsf{SAT}}}$, the operative will then give $\mu$ to the firewall. The firewall will ask on its sanitation interface if $\mu$ should be changed to some $\mu'$. Otherwise let $\mu' = \mu$.

- The firewall asks the core to sign $\mu'$ to get $\sigma$. The firewall might re-randomize $\sigma$ into $\sigma'$. If not let $\sigma' = \sigma$. The firewall hands $(\mu', \sigma')$ to the operative.

- The operative sends $(\mathsf{P}_i, \mu', \sigma')$ via $\mathcal{G}$.

- On receiving $(\mathsf{P}_j, \mu, \sigma)$ from $\mathcal{G}$, the operative drops the message if $\sigma$ is not a signature on $\mu$ by $\mathsf{P}_j$. Otherwise, it hands $(\mathsf{P}_j, \mu)$ to $\mathsf{F}_i$ which asks on its interface $\mathsf{S}_i$ if $\mu$ should be changed to $\mu'$. If not let $\mu' = \mu$. It hands $\mu'$ back to the operative which outputs $(\mathsf{P}_j, \mu')$ on $\mathsf{IO}_i^{\mathcal{F}_{\mathsf{SAT}}}$.

The above is merely meant as a justification that $\mathcal{F}_{\mathsf{SAT}}$ can be implemented in a natural model given for instance subversion-resilient signatures. No formal claims are being made, and the three-tier model would obviously need more details worked out to prove anything formally. The aim of the present paper is to assume that $\mathcal{F}_{\mathsf{SAT}}$ has somehow already been implemented, and then study what other tasks can be securely implemented in the two-tier model.

Note that, in the three-tier model, proving completeness results is trivial as we could in principle let the operative run the protocol alone without talking to a core or firewall. This makes the model somewhat less interesting than the two-tier model. One motivation for studying the two-tier model is that it allows no "easy way out". It only has two components, the firewall and the core. The firewall can be semi-honestly corrupted and the core can be specious, so there is *a priori* no safe place for secrets to hide.

# B Sanitizable Commit & Prove

## B.1 Ingredients

### B.1.1 Malleable Dual-Mode Commitments

A malleable dual-mode commitment scheme consists of the following polynomial-time algorithms $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Com}, \mathsf{Ext}, \mathsf{TCom}, \mathsf{TOpen}, \mathsf{MaulCom}, \mathsf{MaulOpen})$. The probabilistic setup algorithm $\mathsf{Setup}$ takes as input the security parameter $\lambda$ and outputs the setup parameters $\mathsf{par}$. Depending upon the mode (either extraction mode or equivocation mode), the key generation algorithm $\mathsf{KGen}$ can be split into two parts: (i) $\mathsf{KGen}_0$, which on input $\mathsf{par}$ generates a commitment key $\mathsf{ck}$ along with an extraction trapdoor key $\mathsf{extk}$; and (ii) $\mathsf{KGen}_1$, which on input $\mathsf{par}$ generates a commitment key $\mathsf{ck}$ along with a equivocation trapdoor key $\mathsf{tk}$. When the context is clear, we will just write $\mathsf{KeyGen}_b$ as $\mathsf{KeyGen}$. For simplicity, we assume the message space to be $\{0,1\}^*$, and the randomness space to be $\{0,1\}^\lambda$. The algorithm $\mathsf{Com}$ takes as input the commitment key $\mathsf{ck}$, a message $x \in \{0,1\}^*$, and "encodes" $x$ to produce a commitment string $c$ in the commitment space.

Additionally, we require the commitment scheme to satisfy *equivocability*, *extractability* and *malleability* properties, as specified below. The equivocability property ensures that, given the trapdoor key $\mathsf{tk}$, it is possible to open the commitment $c$ to any message. For this purpose, one can use the algorithms $\mathsf{TCom}$ and $\mathsf{TOpen}$. In particular, $\mathsf{TCom}$ takes the trapdoor key $\mathsf{tk}$ as input and produces an equivocal commitment $c$ and an equivocation key $\mathsf{ek}$; on the other hand, $\mathsf{TOpen}$ upon input $\mathsf{ek}$, $c$, and a message $x$ creates an opening $\rho$ of the commitment, so that $c = \mathsf{Com}_{\mathsf{ck}}(x; \rho)$. Extractability requires that, as long as the commitment $c$ is valid, the PPT algorithm $\mathsf{Ext}$ can extract the underlying message $x$ given the extraction key $\mathsf{extk}$. Finally, malleability requires that, given a commitment $c$ corresponding to a message $x$, one can maul it (using algorithm $\mathsf{MaulCom}$) to output a new commitment $\hat{c}$ to a related message $x \oplus r$, for some given $r$. Further, given the decommitment $(x, \rho)$ corresponding to $c$ one can output (using algorithm $\mathsf{MaulOpen}$) a related decommitment $(x \oplus r, \hat{\rho})$ such that $\hat{c}$ can be explained as a commitment to $x \oplus r$ using randomness $\hat{\rho}$.

- *Key Generation.* One can efficiently generate $\mathsf{par}$ by running the setup algorithm $\mathsf{Setup}(1^\lambda)$. Given $\mathsf{par}$, one can efficiently generate a commitment key $\mathsf{ck}$ along with a random extraction trapdoor $\mathsf{extk}$ running the algorithm $\mathsf{KeyGen}_0$. Given $\mathsf{par}$, one can also efficiently generate an equivocable key $\mathsf{ck}'$ along with a trapdoor $\mathsf{tk}$ running the algorithm $\mathsf{KeyGen}_1$.

- *Key Indistinguishability.* Key indistinguishability requires that random extraction keys $\mathsf{ck}$ and equivocation keys $\mathsf{ck}'$ are both *computationally indistinguishable* from random keys, as long as the corresponding trapdoors are not known. In other words, the first outputs of $\mathsf{KeyGen}_0$ and $\mathsf{KeyGen}_1$ are computationally indistinguishable.

- *Equivocability.* For all PPT stateful adversaries $\mathcal{A}$, we have:

$$\mathbb{P}\left[\mathcal{A}(c,\rho)=1 : \begin{array}{c} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda); (\mathsf{ck},\mathsf{tk}) \leftarrow \mathsf{KGen}(\mathsf{par}); \\ x \leftarrow \mathcal{A}(\mathsf{par},\mathsf{ck}); \rho \leftarrow \{0,1\}^\lambda; c := \mathsf{Com}_{\mathsf{ck}}(x;\rho) \end{array}\right]$$
$$\approx_c \mathbb{P}\left[\mathcal{A}(c,\rho)=1 : \begin{array}{c} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda); (\mathsf{ck},\mathsf{tk}) \leftarrow \mathsf{KGen}(\mathsf{par}); x \leftarrow \mathcal{A}(\mathsf{par},\mathsf{ck}); \\ (c,\mathsf{ek}) \leftarrow \mathsf{TCom}_{\mathsf{ck}}(\mathsf{tk}); \rho \leftarrow \mathsf{TOpen}_{\mathsf{ek}}(x,c) \end{array}\right].$$

- *Extractability.* For all PPT stateful adversaries $\mathcal{A}$, we have:

$$\mathbb{P}\left[c \neq \mathsf{Com}_{\mathsf{ck}}(x;\rho) : \begin{array}{c} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda); (\mathsf{ck},\mathsf{extk}) \leftarrow \mathsf{KGen}(\mathsf{par}); \\ c \leftarrow \mathcal{A}(\mathsf{par},\mathsf{ck}); (x,\rho) \leftarrow \mathsf{Ext}(\mathsf{extk},c) \end{array}\right] \leq \mathrm{negl}(\lambda).$$

- *Malleability.* The commitment scheme is *malleable* if the following holds: (i) given $c \leftarrow \mathsf{Com}_{\mathsf{ck}}(x;\rho)$, input $r$ and randomness $\rho'$, there exists an algorithm $\mathsf{MaulCom}$ such that $\mathsf{MaulCom}_{\mathsf{ck}}((c,r);\rho')$ outputs a commitment $\hat{c}$ that is uniformly distributed in the set $\{c : c \leftarrow \mathsf{Com}_{\mathsf{ck}}(x \oplus r)\}$; and (ii) given $(x,\rho)$, input $r$ and randomness $\rho'$, algorithm $\mathsf{MaulOpen}((x,\rho),r,\rho')$ outputs randomness $\hat{\rho}$ such that $\hat{c} := \mathsf{Com}_{\mathsf{ck}}((x \oplus r);\hat{\rho})$

*Remark* 2. Note that the extractability property above implies that the commitment scheme is perfectly binding. On the other hand, the equivocability property above implies that the commitment scheme is computationally hiding. This is because a well-formed commitment is computationally indistinguishable from an equivocal commitment, that can later be opened to any message.

**Instantiation.** It is easy to see that the DDH-based commitment scheme from Section 3.2 satisfies all the above requirements defining a malleable dual-mode commitment scheme.

### B.1.2 Re-randomizable NIZK Arguments

A *re-randomizable NIZK argument system* for a language $L$, associated with an NP-relation $R$, consists of four (probabilistic) polynomial-time algorithms $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{RProof})$ such that the following conditions hold:

- *Completeness.* For all $\mathsf{crs} \in \mathsf{CRSGen}(1^\lambda)$, and $(y,x) \in R$, it holds that $\mathsf{Ver}(\mathsf{crs},y,\pi) = 1$ with probability one over the choice of $\pi \leftarrow \mathsf{P}(\mathsf{crs},y,x)$.

- *Adaptive Soundness.* For all PPT malicious provers $\mathcal{A}$, we have that the following is negligible:

$$\mathbb{P}\Big[\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda); (y, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) : \mathsf{Ver}(\mathsf{crs}, y, \pi) = 1 \text{ and } y \notin L\Big].$$

- *Re-randomizability.* For all PPT adversaries $\mathcal{A}$, the probability of the event $b' = b$ (where $b \in \{0,1\}$ is sampled uniformly at random) in the following game is at most $1/2 + \text{negl}(\lambda)$:
  - $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$
  - $(y, x, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$
  - If $\mathsf{Ver}(\mathsf{crs}, y, \pi) = 0$, or $(y, x) \notin R$, output $\perp$. Otherwise let

$$\pi' \leftarrow \begin{cases} \mathsf{Prove}(\mathsf{crs}, y, x) & \text{if } b = 0 \\ \mathsf{RProof}(\mathsf{crs}, y, \pi) & \text{if } b = 1 \end{cases}$$

  - $b' \leftarrow \mathcal{A}(\mathsf{crs}, \pi')$

- *Adaptive multi-theorem zero-knowledge.* There exists a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ that satisfies the following. For all stateful PPT adversaries $\mathcal{A}$ that only send to its oracle queries $(y, x)$ such $(y, x) \in R$, we have that the following is negligible:

$$\big|\mathbb{P}[\textsc{Real}_{\mathcal{A}}(\lambda) = 1] - \mathbb{P}[\textsc{Simu}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]\big|,$$

where the experiments $\textsc{Real}_{\mathcal{A}}(\lambda)$ and $\textsc{Simu}_{\mathcal{A},\mathcal{S}}(\lambda)$ are defined below:

| $\underline{\textsc{Real}_{\mathcal{A}}(\lambda)}$ | $\underline{\textsc{Simu}_{\mathcal{A},\mathcal{S}}(\lambda)}$ |
|---|---|
| – $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$ | – $(\mathsf{crs}, \mathsf{st}) \leftarrow \mathcal{S}_1(1^\lambda)$ |
| – $b' \leftarrow \mathcal{A}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs})$ | – $b' \leftarrow \mathcal{A}^{\mathcal{S}_2(\mathsf{st},\cdot,\cdot)}(\mathsf{crs})$ |
| – Return $b'$ | – Return $b'$ |

## B.2 The Sanitizing Commit & Prove Protocol

We are now ready to describe our protocol $\widehat{\Pi}_{\mathsf{C\&P}}$ and show that it srUC-realizes the $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$ functionality.

**Protocol overview.** The protocol $\widehat{\Pi}_{\mathsf{C\&P}}$ uses a malleable dual-mode commitment $\Pi_{\mathsf{COM}}$, and a re-randomizable NIZK argument system $\Pi_{\mathsf{NIZK}}$ for proving statements related to the committed values. The protocol is run between a party $\mathsf{P}_i = (\mathsf{C}_i, \mathsf{F}_i)$ that acts as the committer/prover, and parties $\mathsf{P}_{j \neq i}$ that act as verifiers. The party $\mathsf{P}_i$ is allowed to commit to multiple values during a session of the protocol, and then can prove a statement about the list of all values previously committed within that session. The core $\mathsf{C}_i$ takes a message $x$ as input and produces a commitment $c$ by running the $\mathsf{Com}$ algorithm of the $\Pi_{\mathsf{COM}}$ scheme with randomness $\rho$; then, this commitment is delivered to the firewall $\mathsf{F}_i$, that mauls the commitment $c$ and creates $\hat{c}$, *i.e.* a commitment to message $x \oplus r$, using algorithm $\mathsf{MaulCom}$. The sanitized commitment $\hat{c}$ is then sent to all the verifiers $\mathsf{P}_{j \neq i}$. Additionally, the firewall sends back to $\mathsf{C}_i$ the randomness $\rho'$ used to maul the commitment and the blinding factor $r$. The core $\mathsf{C}_i$ can now compute the opening value $\hat{\rho}$ of the mauled commitment $\hat{c}$, using algorithm $\mathsf{MaulOpen}$, and re-compute $\hat{c}$ itself. All those values are saved in lists.

When the core $\mathsf{C}_i$ wants to prove a statement $y$, where the witnesses for $y$ are in the list of committed values, $\mathsf{C}_i$ produces a proof $\pi$ for the statement $(y, \overline{\hat{c}})$, where $\overline{\hat{c}}$ is the list of all (mauled) commitments produced in the current session. The witnesses for producing this proof are the messages contained inside each commitment $\hat{c}$ in the list, namely the xor of the messages $x$ and $r$ in the lists $\overline{x}$ and $\overline{r}$, respectively. The proof $\pi$ and statement $(y, \overline{\hat{c}})$ are delivered to the firewall $\mathsf{F}_i$ that first checks if the list $\overline{\hat{c}}$ in the statement matches the list of mauled commitments it produced previously in the current session, and then checks the validity of the proof $\pi$ against the statement received. If both checks pass, $\mathsf{F}_i$ re-randomizes the proof $\pi$ into $\hat{\pi}$ to remove any possible bias in the distribution of $\pi$. The sanitized proof $\hat{\pi}$ is then sent to all the verifiers $\mathsf{P}_{j \neq i}$.

Note that, in contrast to previous works that built reverse firewalls for MPC [CDN20], we dispense the need of controlled malleability for the NIZK. The reason is that in our protocol the core $\mathsf{C}_i$ learns from $\mathsf{F}_i$ the blinding factor $r$ and the randomness $\rho'$ used to maul the commitment $c$. Hence, $\mathsf{C}_i$ can already produce the proof $\pi$ for the correct statement, *i.e.*, the mauled commitments $\overline{\hat{c}}$ (and not $c$). Thus, it is not necessary for $\mathsf{F}_i$ to maul the proof $\pi$ for a different statement.

### **Protocol** $\widehat{\Pi}_{\mathsf{C\&P}}$ (Realizing Sanitizable Commit and Prove)

Let $\Pi_{\mathsf{COM}}$ be a malleable dual-mode commitment scheme, and let $\Pi_{\mathsf{NIZK}}$ be a re-randomizable NIZK argument system for the relation $R'$ (defined below). Let $R$ be the relation parameterizing the sanitizable commit-and-prove functionality. Then, $R'$ is defined as

$$R' = \{((y, \overline{c}), (\overline{x}, \overline{\rho})) : \forall i, c_i = \mathsf{Com}(x_i; \rho_i) \wedge R(y, \overline{x}) = 1\}\,.$$

The protocol is executed between parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, each consisting of a core $\mathsf{C}_i$ and a firewall $\mathsf{F}_i$. W.l.o.g, let, party $\mathsf{P}_i = (\mathsf{C}_i, \mathsf{F}_i)$ be the committer & prover and all other parties $\mathsf{P}_{j \neq i}$ act as verifiers.

**Setup.** Generate public parameters and keys for the commitment and NIZK argument system.

1. *Commitment scheme:* All parties are given public parameters $\mathsf{par}$ and the commitment key $\mathsf{ck}$ corresponding to $\Pi_{\mathsf{COM}}$, where $\mathsf{par}$ and $\mathsf{ck}$ are obtained by running the $\mathsf{Setup}$ and $\mathsf{KGen}_0$ algorithms of $\Pi_{\mathsf{COM}}$ respectively.
2. *NIZK:* All parties are given $\mathsf{crs}$ for the NIZK argument system, which is obtained by running the algorithm $\mathsf{CRSGen}$ of $\Pi_{\mathsf{NIZK}}$.

**Commitment phase.** In order to generate a commitment, the following steps are performed.

1. *Commit:* Upon input a string $x \in \{0,1\}^\lambda$, the core $\mathsf{C}_i$ commits to $x$ by sampling $\rho \leftarrow \{0,1\}^\lambda$ and computing a commitment $c = \mathsf{Com}_{\mathsf{ck}}(x; \rho)$. The commitment $c$ is then forwarded to the firewall $\mathsf{F}_i$. The input $x$ and the randomness $\rho$ are saved in lists $\overline{x}$, $\overline{\rho}$ respectively (that are initially empty).
2. *Sanitization of commitment:* Upon input a commitment $c$ and a blinding factor $r \in \{0,1\}^\lambda$, the firewall $\mathsf{F}_i$ performs the sanitization of the commitment $c$ as follows: It samples $\rho' \leftarrow \{0,1\}^\lambda$ and computes a mauled commitment $\hat{c} \leftarrow \mathsf{MaulCom}_{\mathsf{ck}}(c, r; \rho')$. The commitment $\hat{c}$ is then sent to all parties $\mathsf{P}_{j \neq i}$. The values $r$ and $\rho'$ are returned to $\mathsf{C}_i$ and saved in lists $\overline{r}$ and $\overline{\rho}'$ respectively (that are initially empty). The sanitized commitment $\hat{c}$ is saved by all parties $\mathsf{P}_{j \neq i}$ in a list $\overline{\hat{c}}$ (that is initially empty).

**Proving phase.** In order to generate a proof, the following steps are performed.

1. *Proving statement:* Upon input a statement $y$:
   - The core $\mathsf{C}_i$ computes new lists $\overline{\hat{\rho}}$ and $\overline{\hat{c}}$ as follows. Let $\ell$ be the size of all the lists maintained by $\mathsf{C}_i$ until now. For all $k \in [\ell]$, the core computes $\overline{\hat{\rho}}[k] \leftarrow \mathsf{MaulOpen}_{\mathsf{ck}}((\overline{x}[k], \overline{\rho}[k]), (\overline{r}[k], \overline{\rho}'[k]))$ and $\overline{\hat{c}}[k] \leftarrow \mathsf{Com}_{\mathsf{ck}}((\overline{x}[k] \oplus \overline{r}[k]); \overline{\hat{\rho}}[k])$, where $\overline{\rho}[k]$ denotes the $k$-th item in the list $\overline{\rho}$.
   - Finally, $\mathsf{C}_i$ computes the proof $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, (y, \overline{\hat{c}}), ((\overline{x} \oplus \overline{r}), \overline{\hat{\rho}}))$. The statement $(y, \overline{\hat{c}})$ and the proof $\pi$ are forwarded to the firewall $\mathsf{F}_i$.

2. *Sanitization of proof:* On input a statement $(y, \overline{\hat{c}})$ and a proof $\pi$, the firewall $\mathsf{F}_i$ first checks if $\mathsf{V}(\mathsf{crs}, (y, \overline{\hat{c}}), \pi) = 1$ and if the list $\overline{\hat{c}}$ is equal to the list of mauled commitments produced so far: If yes, then it proceeds to sanitize the proof by computing $\hat{\pi} \leftarrow \mathsf{RProof}(\mathsf{crs}, (y, \overline{\hat{c}}_i), \pi)$, and it outputs $\hat{\pi}$ to all parties $\mathsf{P}_{j \neq i}$. Otherwise the firewall $\mathsf{F}_i$ ignores the message from $\mathsf{C}_i$.
3. *Verification of proof:* Upon input a statement $(y, \overline{\hat{c}})$ and a proof $\hat{\pi}$, a core $\mathsf{C}_{j \neq i}$ checks if $\mathsf{V}(\mathsf{crs}, (y, \overline{\hat{c}}), \hat{\pi}) = 1$ and if the list $\overline{\hat{c}}$ is equal to the list of mauled commitments received so far: If yes, then the core $\mathsf{C}_{j \neq i}$ accepts the proof, otherwise it rejects it.

**Theorem 5.** *The sanitizing protocol $\widehat{\Pi}_{\mathsf{C\&P}}$ srUC-realizes the $\widehat{\mathcal{F}}_{\mathsf{C\&P}}$ functionality in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model in the presence of up to $n-1$ static malicious corruptions.*

*Proof.* To simplify notation, let $\widehat{\Pi} := \widehat{\Pi}_{\mathsf{C\&P}}$ and $\widehat{\mathcal{F}} := \widehat{\mathcal{F}}_{\mathsf{C\&P}}$. Recall that by definition of subversion resilience, we need to show that $\widehat{\Pi}$ UC-realizes $\widehat{\mathcal{F}}$ in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model, and $\widehat{\mathcal{F}}$ can be written as a well-formed sanitizing ideal functionality. Towards this, we first build a simulator (communicating with $\widehat{\mathcal{F}}$) that simulates an execution of $\widehat{\Pi}$ for the case where $n-1$ parties are malicious, and the remaining party has an honest core and a semi-honest firewall. Note that, strictly speaking, one should also prove security for the case where there are less than $n-1$ malicious corruptions. It is, however, easy to see that proving the case with maximal corruption is complete in the present case. When the commiter/prover is corrupted, then $\widehat{\mathcal{F}}$ gives the simulator the same powers no matter how many verifiers are corrupted, so assuming full corruption gives the adversary more powers (without giving the simulator more powers). If the prover is malicious we are simulating a non-malicious verifier. Since they all act independently, they can all be simulated as we describe next.

**Lemma 9.** *For every malicious adversary $\mathcal{A}$ corrupting $n-1$ parties maliciously and the firewall of the remaining honest party semi-honestly in an execution of the protocol $\widehat{\Pi}$ in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{E}$:*

$$\textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}}_{\widehat{\Pi},\mathcal{A},\mathcal{E}} \equiv \textsc{Exec}_{\widehat{\mathcal{F}},\mathcal{S},\mathcal{E}}.$$

*Proof.* In what follows, we let $j \in [n]$ be the index corresponding to the only party with an honest core $\mathsf{C}_j$ and semi-honest firewall $\mathsf{F}_j$. We consider two cases (depending on whether the committer/prover is maliciously corrupted or not):

**Case 1: Malicious committer/prover.** This corresponds to the case when the core $\mathsf{C}_j$ is one of the (honest) verifiers in the protocol $\widehat{\Pi}$. In this case the simulation proceeds as follows.

**Setup:** The simulator $\mathcal{S}$ runs the $\mathsf{Setup}$ and $\mathsf{KGen}$ algorithms of $\Pi_{\mathsf{COM}}$ to get the public parameters $\mathsf{par}$ and the commitment key $\mathsf{ck}$ and extraction key $\mathsf{extk}$ for $\Pi_{\mathsf{COM}}$. Additionally, $\mathcal{S}$ runs the algorithm $\mathsf{CRSGen}$ of $\Pi_{\mathsf{NIZK}}$ to obtain the $\mathsf{crs}$ for the NIZK scheme.

**Commitment phase:** Here, the adversary commits to one or more witnesses on behalf of the committer.

- Upon receiving a commitment $\hat{c}$ from $\mathcal{A}$, the simulator $\mathcal{S}$ can extract the input $\hat{x} = x \oplus r$ from the commitment by computing $\hat{x} = \mathsf{Ext}(\mathsf{extk}, \hat{c})$. The simulator $\mathcal{S}$ saves $\hat{c}$ in a list of received commitments $\bar{c}$ (that is initially empty).
- The simulator $\mathcal{S}$ then invokes $\widehat{\mathcal{F}}$ with $\hat{x}$ as the input of the core $\mathsf{C}^*$ by sending $(\textsc{Commit}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}^*, \hat{x})$ to $\widehat{\mathcal{F}}$. Additionally, $\mathcal{S}$ also sends the message $(\textsc{Blind}, \mathsf{sid}, \mathsf{cid}, \mathsf{F}^*, 0^\lambda)$ to $\widehat{\mathcal{F}}$ as the input of the firewall $\mathsf{F}^*$. Note that since $\mathsf{C}^*$ is malicious, the functionality $\widehat{\mathcal{F}}$ will send the message $(\textsc{Blinded}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}^*, 0^\lambda)$ to the simulator $\mathcal{S}$.

**Proving phase:** Here, the adversary proves a statement on behalf of the prover.

- Upon receiving a statement $(y, \bar{\bar{c}})$ and a proof $\hat{\pi}$ from $\mathcal{A}$, the simulator $\mathcal{S}$ sends the message $(\textsc{Prove}, \mathsf{sid}, \mathsf{C}^*, y)$ to $\widehat{\mathcal{F}}$ on behalf of $\mathsf{C}^*$.
- Upon message $(\textsc{Sanitize}, \mathsf{sid}, \mathsf{C}^*, y)$ from $\widehat{\mathcal{F}}$, the simulator $\mathcal{S}$ sends the message $(\textsc{Continue}, \mathsf{sid}, \mathsf{C}^*, y)$ to $\widehat{\mathcal{F}}$ on behalf of $\mathsf{F}^*$.

Note that the setup phase is perfectly simulated, as the CRS and the commitment parameters are generated honestly (although the simulator keeps the corresponding extraction trapdoor). The commitment phase is also perfectly simulated, unless the algorithm $\mathsf{Ext}$ fails to extract the correct input $\hat{x}$ from the commitment.; the latter, however, happens with negligible probability only. Finally, observe that during the proving phase, the simulation only fails if the proof $\hat{\pi}$ sent by the adversary is a valid proof of a false statement $(y, \bar{\bar{c}})$; a standard reduction to the soundness property of the NIZK argument system shows that this only happens with negligible probability.

**Case 2: Honest commiter/prover.** This corresponds to the case when the core $\mathsf{C}_j$ is the (honest) committer/prover and $\mathsf{F}_j$ is the semi-honest firewall in the protocol $\widehat{\Pi}$. In this case the simulation proceeds as follows.

**Setup:** The simulator $\mathcal{S}$ runs the $\mathsf{Setup}$ and $\mathsf{KGen}_1$ algorithms of $\Pi_{\mathsf{COM}}$ to get the public parameters $\mathsf{par}$, the commitment key $\mathsf{ck}$, and trapdoor key $\mathsf{tk}$ for $\Pi_{\mathsf{COM}}$. Additionally, $\mathcal{S}$ runs the algorithm $\mathsf{CRSGen}$ of $\Pi_{\mathsf{NIZK}}$ to obtain the $\mathsf{crs}$ for the NIZK scheme and a trapdoor $\mathsf{st}$ for simulating proofs.

**Commitment phase:** Here, the simulator must fake a commitment sent by the honest committer.

- Upon receiving the message $(\textsc{Receipt}, \mathsf{sid}, \mathsf{cid}, \mathsf{C}_j)$ from $\widehat{\mathcal{F}}$, the simulator computes an equivocable commitment $(c, \mathsf{ek}) \leftarrow \mathsf{TCom}_{\mathsf{ck}}(\mathsf{tk})$.
- Since the firewall $\mathsf{F}_j$ is semi-honest, the simulator $\mathcal{S}$ knows the input string $r \in \{0,1\}^\lambda$ of $\mathsf{F}_j$. Hence, the simulator $\mathcal{S}$ can sanitize the commitment $c$ by computing $\hat{c} = \mathsf{MaulCom}_{\mathsf{ck}}(c, r; \rho')$ for a random $\rho' \in \{0,1\}^\lambda$. The simulator $\mathcal{S}$ then sends $\hat{c}$ to all parties $\mathsf{P}_{j \neq i}$, and saves $\hat{c}$ in a list $\bar{\bar{c}}$.

**Proving phase:** Here, the simulator must fake a proof sent by the honest prover.

- Upon receiving the message $(\textsc{Proved}, \mathsf{sid}, \mathsf{C}_i, y)$ from $\widehat{\mathcal{F}}$, the simulator $\mathcal{S}$ can simulate a valid proof $\pi$ for statement $(y, \bar{\bar{c}})$ by running the simulator of the NIZK scheme with trapdoor $\mathsf{st}$.

To show that the simulation above is indistinguishable from the real world, we define a series of hybrids. We start with $\textsc{Hyb}_0$ that is the ideal world, and from there define a few intermediate hybrids that are exactly the same as the previous, except for the changes described below. Finally, we end with $\textsc{Hyb}_3$ which is the real world. We then argue that adjacent hybrids are indistinguishable.

**Hybrid 0:** This is the ideal world with the above described simulator.

**Hybrid 1:** The simulator $\mathcal{S}$ receives the input $x$ of the honest committer $\mathsf{C}_j$. After producing the equivocable commitment $c$, the simulator immediately equivocates the commitment

$c$ for the input message $x$.[4] Note that the difference between hybrids $\textsc{Hyb}_0$ and $\textsc{Hyb}_1$ is just syntactic, as in both hybrids the equivocable commitment $c$ does not change. Hence,

$$\textsc{Hyb}_0 \equiv \textsc{Hyb}_1.$$

**Hybrid 2:** The simulator $\mathcal{S}$ now produces the real commitment $c$ (instead of an equivocable commitment as in the previous hybrid) to message $x$ by computing $c \leftarrow \mathsf{Com}_{\mathsf{ck}}(x;\rho)$ for a random $\rho \in \{0,1\}^\lambda$. The difference between hybrids $\textsc{Hyb}_1$ and $\textsc{Hyb}_2$ is that in the former the commitment $c$ is produced as an equivocable commitment, and in the latter the commitment $c$ is a real commitment to message $x$. Any adversary with a non-negligible advantage in distinguishing $\textsc{Hyb}_1$ and $\textsc{Hyb}_2$ can be used to build an adversary with a non-negligible advantage in violating the equivocability property of the commitment scheme $\Pi_{\mathsf{COM}}$. Hence,

$$\textsc{Hyb}_1 \approx_c \textsc{Hyb}_2.$$

**Hybrid 3:** The simulator $\mathcal{S}$ now, instead of simulating proofs $\pi$, it runs $\pi \leftarrow \mathsf{P}(\mathsf{crs}, (y, \overline{\tilde{c}}), ((\overline{x} \oplus \overline{r}), \overline{\tilde{\rho}}))$ to produce real proofs for the statement $(y, \overline{\tilde{c}})$. Note that the simulator keeps all lists $\overline{x}, \overline{r}, \overline{\tilde{\rho}}$ and $\overline{\tilde{c}}$. This hybrid is exactly the same as an execution of the real-world protocol. The difference between hybrids $\textsc{Hyb}_2$ and $\textsc{Hyb}_3$ is that in the former the proof $\pi$ is simulated, and in the latter the proof $\pi$ is a real proof. Any adversary with a non-negligible advantage in distinguishing $\textsc{Hyb}_2$ and $\textsc{Hyb}_3$ can be used to build an adversary with a non-negligible advantage in violating the zero-knowledge property of the NIZK argument system $\Pi_{\mathsf{NIZK}}$. Hence,

$$\textsc{Hyb}_2 \approx_c \textsc{Hyb}_3.$$

This finishes the proof. □

Next, we show that the firewall $\mathsf{F}_j$ of the committer/prover is strongly sanitizing (see Definition 10), meaning that a specious core behind the firewall looks like an honest core.

**Lemma 10.** *The firewall $\mathsf{F}_j$ of the committer/prover $\mathsf{C}_j$ in $\widehat{\Pi}$ is strongly sanitizing.*

*Proof.* We need to show that for all poly-time environments $\mathcal{E}$ which do not corrupt the firewall $\mathsf{F}_j$, but which are allowed a specious corruption of the core $\mathsf{C}_j$, it holds that

$$\textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}}_{\widehat{\Pi},\mathcal{A},\mathcal{E}} \approx \textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}}_{\widehat{\Pi}',\mathcal{A},\mathcal{E}},$$

where $\mathcal{A}$ is the dummy adversary and where $\widehat{\Pi}$ and $\widehat{\Pi}'$ run with dummy parties except for $\mathsf{P}_j$ that is either taken to be $(\mathsf{C}_j, \mathsf{F}_j)$ or $(\widehat{\mathsf{C}}_j, \mathsf{F}_j)$ for an incorruptible core $\widehat{\mathsf{C}}_j$. Recall that when an incorruptible core receives a specious corruption $(\textsc{Specious}, \widetilde{\mathsf{C}}_j)$ from the environment, it ignores it and keeps running the code of $\mathsf{C}_j$.

Note that, in a real execution, the honest core $\mathsf{C}_j$ produces a commitment $c$ (to input $x$) that is uniformly distributed in the space of commitments to $x$. Thus, the sanitized commitment $\hat{c}$ produced by the firewall by mauling $c$ with $r \in \{0,1\}^\lambda$ is also uniformly distributed in the space of commitments to $x \oplus r$. Moreover, any commitment $\tilde{c}$ output by a specious core $\widetilde{\mathsf{C}}_j$ must be well-formed, i.e., there must exist an opening $\rho \in \{0,1\}^\lambda$ and a message $x \in \{0,1\}^\lambda$ such that $\tilde{c}$ opens to $x$ using $\rho$, as otherwise we can build a poly-time text $\mathsf{T}$ that tells apart non-rewinding black-box access to either $\widetilde{\mathsf{C}}_j$ or $\mathsf{C}_j$ by asking it to first compute and then open a commitment.

[4] We stress that in the actual simulation, $\mathcal{S}$ can never receive the inputs of honest parties. However, since the hybrids can be seen as "mental experiments", we are allowed to do that.

This shows that a specious core, except with negligible probability, still outputs a well-formed commitment $\tilde{c}$; given such a commitment, the firewall $\mathsf{F}_j$ produces a sanitized committent $\hat{c}$ (by mauling $c$ with $r \in \{0,1\}^\lambda$) that is uniformly random in the space of commitments to the string $x \oplus r$.

Analogously, the honest core $\mathsf{C}_j$ produces a proof $\pi$ for statement $(y, \overline{\hat{c}})$ that is uniformly distributed in the space of proofs for that statement. Thus, the sanitized proof $\hat{\pi}$ produced by the firewall by rerandomizing $\pi$ is still uniformly distributed in the space of proofs for that statement. Moreover, any proof $\tilde{\pi}$ output by a specious core $\widetilde{\mathsf{C}}_j$ must be valid w.r.t the statement $(y, \overline{\hat{c}})$, as otherwise we can build a poly-time text $\mathsf{T}$ that tells apart non-rewinding black-box access to either $\widetilde{\mathsf{C}}_j$ or $\mathsf{C}_j$ by asking it to first compute a proof for $(y, \overline{\hat{c}})$ and then try to verify it. This shows that a specious core, except with negligible probability, still outputs a valid proof $\tilde{c}$; given such a proof, the firewall $\mathsf{F}_j$ produces a sanitized proof $\hat{\pi}$ (by rerandomizing $\pi$) that is uniformly random in the space of proofs for statement $(y, \overline{\hat{c}})$. The lemma follows. □

The theorem statement now follows by looking at the standard corruption transition table used by the well-formed sanitizing ideal functionality $\widehat{\mathcal{F}}$. Since the adversary maliciously corrupts up to $n-1$ verifiers, there is at least one party which is the committer & prover for which either (i) the core is honest and the firewall is semi-honest, or (ii) the core is specious and the firewall is honest. By Lemma 2, since an honest firewall is strongly sanitizing (as shown in Lemma 10), the core in case (ii) can be taken to be honest. Hence, the statement follows directly by Lemma 9. Note that here we are assuming that $\widehat{\mathcal{F}}$ treats a corruption with flavor Isolate as a Malicious corruption. □