Key Exchange in the Post-Snowden Era: UC Secure Subversion-Resilient PAKE

Suvradip Chakraborty[1], Lorenzo Magliocco[2], Bernardo Magri[3], and Daniele Venturi[2]

[1]VISA Research
[2]Sapienza University of Rome
[3]University of Manchester

November 28, 2023

Abstract

Password-Authenticated Key Exchange (PAKE) allows two parties to establish a common high-entropy secret from a possibly low-entropy pre-shared secret such as a password. In this work, we provide the first PAKE protocol with *subversion resilience* in the framework of universal composability (UC), where the latter roughly means that UC security still holds even if one of the two parties is malicious and the honest party's code has been subverted (in an undetectable manner).

We achieve this result by sanitizing the PAKE protocol from oblivious transfer (OT) due to Canetti *et al.* (PKC'12) via cryptographic reverse firewalls in the UC framework (Chakraborty *et al.*, EUROCRYPT'22). This requires new techniques, which help us uncover new cryptographic primitives with sanitation-friendly properties along the way (such as OT, dual-mode cryptosystems, and signature schemes).

As an additional contribution, we delve deeper in the backbone of communication required in the subversion-resilient UC framework, extending it to the *unauthenticated* setting, in line with the work of Barak *et al.* (CRYPTO'05).

Keywords: PAKE · subversion resilience · universal composability

1 Introduction

Authenticated Key Exchange (AKE) allows two parties to generate a shared high-entropy secret and mutually authenticate themselves, by means of identifiers such as public keys, signatures or shared passwords. As such, AKE allows two parties to establish a secure channel. Due to its sensitive nature, malicious actors may have a particular interest in undermining the security of AKE protocols (*e.g.*, by leaking the password of an honest party, or by establishing a shared key without authentication). To this extent, AKE protocols are typically designed in the setting of multi-party computation, where the adversary controls the communication channels and can corrupt some of the parties. Corrupted parties either simply follow the protocol (so-called *semi-honest* corruptions), or deviate arbitrarily from its intended execution (so-called *malicious* corruptions).

This threat model is widely adopted in the literature. However, it relies on the assumption of having access to uncorrupted parties that run the protocol exactly as prescribed. Unfortunately, as shown by the shocking Edward Snowden's revelations, the latter assumption may not hold in practice, as the machine of honest parties could have been compromised in an undetectable manner, both in the case of its hardware (*e.g.*, by means of backdoored components) or its software (*e.g.*, algorithm-substitution attacks, purposefully designed leaky constructions, or mistakenly instantiated protocols). A possible mitigation consists in equipping parties with *cryptographic reverse firewalls* (RFs), as first defined by Mironov and Stephens-Davidowitz [24]. These objects allow to sanitize inbound and outbound messages of the party they are attached to, thus destroying

any potential side-channel while preserving functionality and security of the underlying protocol. The idea here is that protocol designers can instantiate parties and their respective RF on different physical machines on the same local network in order to achieve security in the presence of subversion attacks.

While the original formalism only accounted for standalone security, where each protocol is run in isolation, the setting of RFs has recently been extended to the universal composability (UC) framework by Chakraborty, Magri, Nielsen and Venturi [15]. The latter ensures that subversion resilience holds even under arbitrary composition of the designed protocol with other protocols, without redoing the security analysis from scratch, and thus yielding a modular design of subversion-resilient cryptographic protocols.

1.1 Password-Authenticated Key Exchange

In this work, we focus on instantiating Password-Authenticated Key Exchange (PAKE) in the subversion-resilient UC framework, in which parties can derive a high-entropy secret key by simply verifying their identities by means of a low-entropy common password. Given that passwords are considered to be low-entropy, the security definition of PAKE must take into account the fact that the adversary can guess the password with non-negligible probability. Thus, a protocol realizing PAKE is secure if no adversary is able to break it with probability better than guessing the password. Moreover, the PAKE functionality restricts the ideal adversary to only perform *online* password guesses: in other words, the transcript of a PAKE protocol must not help the adversary to perform a dictionary (*i.e.*, offline) attack.

1.2 Our Results

Our main contribution consists in constructing the first UC PAKE protocol with security in the presence of (specious) subversion attacks, via RFs. Following [15], we consider a setting where each party is split into a core (which has secret inputs and is in charge of generating protocol messages) and a RF (which has no secrets and sanitizes the outgoing/incoming communication from/to the core using random coins). Both the core and RF are subject to different flavours of corruption, modelling different kinds of subversion attacks.

Our PAKE protocol is obtained by sanitizing the UC randomized equality protocol from oblivious transfer (OT) by Canetti *et al.* [10]. As we explain in the next subsection, obtaining this result requires essential changes to the original protocols' design, such as the definition and realization of sanitizable variants of intermediate ideal functionalities, and the introduction of technical tools of independent interest.

One difficulty in the realization of PAKE is that one *cannot* rely on authenticated channels. As shown by Barak *et al.* [5], this difficulty can be tackled generically by first designing a PAKE protocol assuming authenticated channels, and then compiling it into another protocol *without* authenticated channels using the concept of "split functionalities". Such functionalities basically allow the adversary to disconnect parties completely, and engage in separate executions with each one of the two parties, where in each execution the adversary plays the role of the other party. We follow a similar recipe in the design of our PAKE protocol. In particular, we first realize subversion-resilient PAKE assuming the existence of a functionality for sanitizable authenticated communication (which already appeared in [15], and is denoted by $\mathcal{F}_{\mathsf{SAT}}$). Following [5], we then define a weaker split-authenticated variant $s\mathcal{F}_{\mathsf{SAT}}$ that allows the adversary to partition parties, and prove that a modification of the transformation by Barak *et al.* allows to lift any protocol that realizes a functionality $\mathcal{G}$ assuming authenticated channels to one that realizes the corresponding "split version" (*i.e.*, $s\mathcal{G}$) without any assumption on channels, even in the presence of subversion.

Finally, we realize $s\mathcal{F}_{\mathsf{SAT}}$ by sanitizing the protocol due to Barak *et al.* [5], introducing a new notion of *sanitizable* signature schemes with a matching security property. This improves on an open problem from [15], where the authors were only able to realize $\mathcal{F}_{\mathsf{SAT}}$ assuming the presence of a PKI and by moving to a "three-tier model" variant of the framework, in which each party has an additional *operative* component that may only be honest or malicious. Even if used exclusively throughout the setup phase of the protocol, providing access to an operative component that is immune to subversion is a strong assumption that definitely weakens any result achieved in the framework: indeed, the three-tier model provides a trivial solution to counteract specious corruptions of the core for *any* functionality, as the operative is in principle allowed to run any protocol on behalf of the core. On the contrary, we realize the backbone of communication

among components in the two-tier model without assuming a PKI, although only for the unauthenticated setting (*i.e.*, $s\mathcal{F}_{\mathsf{SAT}}$).

1.3 Technical Overview

Below, we provide an overview of the technical contributions, explaining the main ideas and tools behind our subversion-resilient PAKE protocol.

1.3.1 Sanitizing OT.

Defining oblivious transfer in the presence of subversion attacks is a tricky task, as the (non-sanitized) functionality would allow a (specious) receiver to obtain exactly one of the inputs of the sender, which may act as a trigger if sampled maliciously. Similarly, it would allow a (specious) sender to sample the inputs in a leaky manner and send them over to a corrupted party. For this reason, in our sanitizable OT ideal functionality $\mathcal{F}_{\mathsf{sOT}}$ (depicted in Figure 1) both firewalls are allowed to blind the sender's inputs by means of a blinding operation. This way, the sender's firewall can sanitize the sender's randomly chosen inputs, and the receiver's firewall can sanitize the inbound inputs.

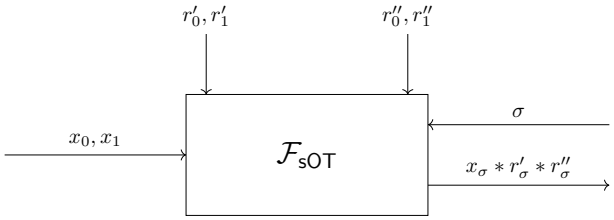

Figure 1: Our sanitizable OT functionality $\mathcal{F}_{\mathsf{sOT}}$. r'_i, r''_i values are sampled from firewalls, and $*$ is an appropriate blinding operation for the input domain.

Here, we introduce a different technique compared to that of the seminal framework. Namely, the functionality allows firewalls to explicitly contribute to the sanitation, and disregard their contribution whenever the overall party related to that firewall is corrupted. From a formal standpoint this is allowed, as there exists a corruption translation table that maps corruptions of individual components of a party to a corruption for the entire party, and currently the srUC framework only supports static corruptions, so the functionality knows in advance which parties are corrupted. This also makes sense for what concerns simulation, as once we have mapped components to a malicious party we shouldn't be simulating anything that occurs within that malicious party. In particular, the notion of blinding may not even be well defined at all, and it suffices to extract from the malicious party only the actual functional inputs to the functionality (*e.g.*, for sanitizable OT it suffices to craft a query for the malicious receiver's input bit, disregarding the blinding factors of its firewall).

In order to instantiate $\mathcal{F}_{\mathsf{sOT}}$, we introduce a new primitive that we call *sanitizable homomorphic* dual-mode cryptosystems, that provides: (1) a procedure to carry out homomorphic operations on ciphertexts (*e.g.*, $\mathsf{Enc}(m_1) * \mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 * m_2)$), (2) a procedure to maul an encryption key pk to a different encryption key $\widetilde{pk}$, and (3) a procedure to maul a ciphertext under encryption key $\widetilde{pk}$ to a ciphertext of the same message under encryption key pk. Item (1) allows firewalls to sanitize the messages input to the OT, and items (2, 3) allow to first blind an inbound public key, introducing a layer of sanitation, and align encryptions accordingly, stripping that layer of sanitation away to preserve correctness. The construction from DDH of Peikert *et al.* [25] can be extended to verify our newly introduced properties in a straight-forward manner.

Finally, we instantiate the functionality by proposing an appropriate sanitation of the protocol of Peikert *et al.* [25], which unfolds as follows. The receiver produces a key pair that may only be used to decrypt values on the encryption branch matching the choice bit σ (out of the two). The key is sanitized twice by firewalls, and the sender encrypts value x_b on encryption branch b. Then, firewalls remove the layers of sanitation appropriately, and the receiver can decrypt only the encryption on branch $b = \sigma$.

In the security proof, we first show that the construction is strongly sanitizing, *i.e.*, a specious core with a honest firewall is indistinguishable from an incorruptible core with a honest firewall, by using the aforementioned properties. After that, the simulation becomes extremely close to the one of the original protocol, leveraging on the two (computationally indistinguishable) modes of the CRS to extract an adversary's inputs and produce a consistent query to $\mathcal{F}_{\mathsf{sOT}}$.

1.3.2 Sanitizing Randomized Equality.

In the RE from OT protocol of Canetti *et al.* [10], for an n-bit password, each party runs $\mathcal{F}_{\mathsf{OT}}$ n-times as the sender, inputting two random strings for each OT run, and n-times as the receiver, inputting the i-th bit of their password. Intuitively, a party can choose the same random strings their peer has received only if the passwords are the same, and these strings can all be combined to derive a common shared key.

After defining $\mathcal{F}_{\mathsf{sOT}}$, we realize the regular randomized equality functionality $\mathcal{F}_{\mathsf{RE}}$ in the subversion setting in a straight-forward manner. In order to thwart information leakage originating from a biased sampling of the strings, as well as inbound input-triggering strings, both firewalls blind the sender's inputs in both OT batches with locally-sampled random strings. The trick to preserve correctness leverages on the symmetrical structure of the protocol: namely, random strings used for the i-th OT in which a core acts as the sender are re-used for the i-th OT in which the same core acts as the receiver.

Interestingly, we remark that *explicit* mutual authentication is not as simple to achieve as in settings without subversion, in which common transformations involve equality testing of a part of the key (or a function of it) suffice to achieve that property (*e.g.*, by locally computing a PRF with key the output key on a fixed value and mutually sending the output to the other party). Sanitizing these kind of transformations comes at the cost of breaking correctness, and leaving them unchecked allows a malicious party to craft a message that would trigger the subversion of the peer in an undetectable manner.

1.3.3 Split functionalities in the srUC model.

A PAKE protocol establishes (over an *unauthenticated* channel) a secret key among parties that share a common password. Thus, it makes little sense to build a PAKE protocol in a setting that already assumes the existence of authenticated channels, as the srUC model does in the form of the "sanitizable authenticated transmission" $\mathcal{F}_{\mathsf{SAT}}$ functionality.

The problem of achieving any form of secure computation (including protocols such as PAKE) in the UC unauthenticated channel setting was first described by Barak *et al.* [5]. In their setting, all the messages sent by parties can be tampered with and manipulated by the adversary unbeknownst to honest parties. The authors show that in this model it is not possible to achieve the same guarantees as with authenticated channels, however meaningful guarantees can still be provided. Namely, the worst the adversary can do is to split honest parties into independent execution sets, where the remaining parties of the protocol are controlled by the adversary. In this way honest parties can run the entire protocol against the adversary without even noticing it, however parties can rest assured that they will complete the entire run of the protocol interacting with the same set of parties since the start, albeit without knowing the actual identities of the parties. In [5] this notion is captured in what the authors call *split functionalities*. They then proceed to show that any UC protocol realizing some functionality $\mathcal{F}$ relying on authenticated channels can be compiled into a similar protocol that realizes the split functionality $s\mathcal{F}$, but now just relying on *unauthenticated* channels.

In this work, we extend the notion of split functionalities to the srUC model by showing how it can be adapted to account for specious environments and adversaries, as required by the srUC model. More specifically, we show that the generic transformation of [5] for split protocols carries over to our setting whenever the underlying *unauthenticated* channel is sanitizable. In particular, the crucial component of the transformation is the construction of a protocol realizing the split version of the $\mathcal{F}_{\mathsf{SAT}}$ functionality of [15], that we call $s\mathcal{F}_{\mathsf{SAT}}$. For that, we introduce a new primitive that we call *sanitizable* signatures that: (1) provides a procedure to maul a verification key vk into $\widetilde{\mathsf{vk}}$, (2) a procedure to maul a signature under verification key $\widetilde{\mathsf{vk}}$ back into a signature under verification key vk, and (3) is equipped with a function f such that $f(\mathsf{vk}_i, \widetilde{\mathsf{vk}}_j) = f(\widetilde{\mathsf{vk}}_i, \mathsf{vk}_j)$, where $\widetilde{\mathsf{vk}}_i$ and $\widetilde{\mathsf{vk}}_j$ being verification keys mauled under the same randomness.

We show that the BLS [8] signature scheme is a sanitizable signature scheme where the function f is a bilinear map. Our protocol allows the firewalls to sanitize an outbound verification key and later re-align outbound signatures accordingly to preserve correctness. The bilinear map is used to allow parties to recompute the same session ID (sid) for the protocol in the presence of firewalls mauling keys. Once this "link initialization" phase is complete, components of parties exchange messages as in $\mathcal{F}_{\mathsf{SAT}}$, except that the adversary may now deliver arbitrary messages to parties within different authentication sets. We note however that our instantiation of $s\mathcal{F}_{\mathsf{SAT}}$ is only for the 2-party setting, as the bilinear map only allows for 2 parties to correctly compute the protocol sid in a consistent manner. This inevitably restricts the transformation to only capture 2-party functionalities in the srUC model.

Once a protocol for $s\mathcal{F}_{\mathsf{SAT}}$ is in place, we simply white-box inspect the proofs of [5] adapting them for specious environments and adversaries as needed. The final result is a theorem stating that any 2-party split functionality can be realized in the srUC model using only *unauthenticated* channels (in the $s\mathcal{F}_{\mathsf{SAT}}$-hybrid model).

1.4 Related Work

Next we discuss related works on the topics of reverse firewalls and subversion-resilient cryptography in general, and on PAKE protocols.

Reverse Firewalls and Subversion. Reverse firewalls were introduced by Mironov and Stephens-Davidowitz [24], who showed how to construct reverse firewalls for oblivious transfer (OT) and two-party computation with semi-honest security. Follow up works showed how to construct reverse firewalls for many other cryptographic primitives and protocols including: secure message transmission and key agreement [18, 16], interactive proof systems [21], and maliciously secure MPC for both the case of static [13] and adaptive [14] corruptions. However, most of these constructions lack modularity, as the security of each firewall is proven in isolation and does not extend to larger protocols when combined with other firewalls. This was addressed by Chakraborty, Magri, Nielsen and Venturi [15] with the proposal of the Subversion-Resilient Composability framework (srUC). The srUC allowed for the first time to build and to analyse subversion-resilient protocols under composition. [15] shows how to sanitize the classical GMW compiler [22] for MPC under subversion. Towards that, it also introduces the concept of sanitizable commitment and sanitizable commit-and-prove.

Additional work on subversion includes algorithm substitution attacks [7, 4, 17], parameter subversion [6, 1, 20, 2], Cliptography [3, 27, 26], and subliminal channels [28, 29] to list a few.

PAKE. The seminal work by Canetti *et al.* [11] formalizes PAKE as an ideal functionality, and proposes an efficient protocol securely realizing this functionality in the setting of malicious corruptions and under *universal composability* [9], *i.e.*, when protocols can be arbitrarily composed with other protocols. The description was later extended to *explicit* mutual authentication in [23] and [10], in which parties are able to tell whether they effectively authenticated or not.

Prior to our work, the only alternative to achieve subversion-resilient PAKE in the UC framework was the sanitized version of the GMW compiler of [15]. However, this solution is highly impractical. Our solution is much more efficient and presents little sanitation overhead, with the drawback of being functionality-specific. We refer the reader to [24, 15] for alternative approaches to counter subversion attacks, such as *watchdogs* and *self-guarding*.

1.4.1 Organization.

In Section 2, we give a concise introduction to the subversion-resilient UC framework of [15]. In Section 3, we define and instantiate sanitizable oblivious transfer. In Section 4, we instantiate a sanitized protocol for the randomized equality ideal functionality. In Section 5, we define and instantiate the sanitizable split-authenticated functionality, and port the transformation of Barak *et al.* [5] that allows to remove authenticated channels in our framework. In Section 6, we combine the results of previous sections to

achieve PAKE. Finally, in Section 7, we conclude the paper with a few related open problems for further research.

2 A Brief Recap of Subversion-resilient UC

We give a brief overview of the definition of subversion resilience in the UC framework (srUC for short). We refer the reader to [15] for a complete treatment of the model, and to Appendix A for a brief introduction to the UC framework [9].

2.1 Corruption Types

Each party P_i in the protocol is modelled as two independent parties: a core C_i, which hosts the code associated with the protocol (and may contain secrets), and a firewall F_i, which may intervene on all the messages associated with their respective core (both inbound and outbound). Since cores and firewalls are independent parties, they may also be corrupted independently. The model of [15] specifies that the relevant corruption cases for the core are Honest, Malicious or Specious, while the ones for the firewall are Honest, SemiHonest or Malicious. Mapping the corruption possibilities for the parties $\mathsf{P}_i = (\mathsf{C}_i, \mathsf{F}_i)$ in a regular UC functionality gives rise to the following corruption table:

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

Specious corruption. A specious corruption is a type of subversion where the subverted core looks indistinguishable from the honest core to any efficient test. The main idea is that we consider corruptions where a core C_i has been replaced by another implementation $\widetilde{\mathsf{C}}_i$ which cannot be distinguished from C_i by black-box access to $\widetilde{\mathsf{C}}_i$ or C_i. Intuitively, a specious corruption can be thought of as a subversion that remains undetectable.

Isolate corruption. Isolate is a weaker type of corruption that models the setting where a malicious firewall simply cuts the communication of an honest core with the outside world. This is typically modelled as a Malicious corruption and can be safely dropped from the analysis.

Strong sanitation. A firewall is strongly sanitizing if an adversary is unable to distinguish an execution of the protocol with a specious core equipped with an honest firewall from an execution of the protocol with an honest core equipped with an honest firewall. As shown in [15], whenever the firewalls are strongly sanitizing, the Specious core and Honest firewall case boils down to an Honest party in the functionality.

2.2 Ideal Functionalities

There are two types of ideal functionalities in srUC: *sanitizable* functionalities and *regular* functionalities. Sanitizable functionalities are the ones where cores and firewalls explicitly interact with the functionality. For that, sanitizable functionalities expose, for each party P_i, an input-output interface IO_i that interacts with the core C_i, and a sanitation interface S_i that interacts with the firewall F_i. Regular functionalities have the same flavor of the functionalities used in the UC framework, where the functionality will only communicate with parties and is not aware of cores and firewalls. The goal of considering regular functionalities is that it is perfectly valid and desirable to be able to build protocols that realize a regular functionality (*e.g.,* coin tossing) under subversion attacks. However, since there is no support for sanitation interfaces in

regular functionalities, the model considers a wrapped version of the functionality $\mathcal{F}$ denoted by $\mathsf{Wrap}(\mathcal{F})$, that handles all the boilerplate code of translating the combinations of corruptions of cores and firewalls to corruptions of parties in $\mathcal{F}$. The wrapper also passes any message coming from the functionality and directed to party P_i, to the corresponding core C_i and firewall F_i, and it is needed to avoid trivial distinguishing attacks in the UC framework, since the actual protocol will be implemented with cores and firewalls.

The definition below captures security of a *sanitizing* protocol[1] Π that instantiates cores and firewalls interacting with the interfaces exposed by a sanitizable ideal functionality $\mathcal{G}$ in order to UC-realize a wrapped regular functionality $\mathsf{Wrap}(\mathcal{G})$ in the $\mathcal{G}$-hybrid model. This yields so-called *wrapped* subversion-resilient UC security (wsrUC).

Definition 1 (Wrapped subversion-resilient UC security [15])**.** *Let $\mathcal{F}$ be an ideal functionality for n parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Let Π 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 the sanitizing protocol Π. We say that Π wsrUC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if Π 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.*

For completeness, we also report the more general definition of srUC security, as we will instantiate both wrapped functionalities and sanitizable functionalities.

Definition 2 (Subversion-resilient UC security [15])**.** *Let $\mathcal{F}$ be an ideal functionality for n parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Let Π 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 a protocol Π interacting on IO interfaces and sanitation interfaces. We say that Π srUC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if $\mathcal{F}$ can be written as a well-formed sanitizable ideal functionality[2], and Π UC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, with the restriction that we only quantify over specious environments and specious adversaries.*

2.3 Communication channels

In all protocols in [15], communication is mediated by means of a sanitizable ideal functionality for authenticated communication $\mathcal{F}_{\mathsf{SAT}}$, which fundamentally includes three behaviours:

- It allows to distribute a setup (e.g., a CRS) by means of a Setup algorithm.
- It provides *secure* channels between cores and their respective firewall.
- It provides *authenticated* channels between firewalls.

We report a variant of the description of $\mathcal{F}_{\mathsf{SAT}}$ in what follows that does *not* include the possibility for distributing a setup. This is a design choice that allows to better separate the component for setup from the component of communication, since not all protocols may require both components at once. The former may be represented by means of a separate ideal functionality $\mathcal{F}_{\mathsf{crs}}$.

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

- On input $(\textsc{Send}, a, \mathsf{P}_j)$ on IO_i, it forwards the tuple on S_i. As in the original description, we assume that a is sent at most once from honest parties.
- On input $(\textsc{Send}, b, \mathsf{P}_k)$ on S_i, it leaks the tuple to the adversary $\mathcal{S}$, along with its sender P_i, and internally stores the tuple.
- On input $(\textsc{Deliver}, (\textsc{Send}, \mathsf{P}_i, b, \mathsf{P}_k))$ from the adversary, where the $\textsc{Send}$ tuple is stored, it outputs $(\textsc{Receive}, \mathsf{P}_i, b)$ on S_k and deletes the tuple.
- On input $(\textsc{Receive}, \mathsf{P}_m, c)$ on S_k, it outputs $(\textsc{Receive}, \mathsf{P}_m, c)$ on IO_k.

[1] We use the terms "sanitizing protocol" and "sanitized protocol" interchangeably to refer to subversion-resilient protocols realizing a regular UC functionality.

[2] A well-formed sanitizable ideal functionality can be decomposed into a central part that does the computation and interacts with parties, and an outer layer exposing IO and sanitation interfaces, similarly to a wrapped ideal functionality.

Our protocols in Sections 3 and 4 assume this functionality as the backbone for communication. Unfortunately, this definition induces a core-to-core authenticated channel between parties, making any instantiation for PAKE relying on $\mathcal{F}_{\mathsf{SAT}}$ redundant. In Section 5, we overcome this limitation by defining a weaker functionality $s\mathcal{F}_{\mathsf{SAT}}$ that allows the adversary to partition parties, in line with the work of Barak et al. [5].

3 Sanitizing Oblivious Transfer

In this section, we first propose a sanitizable ideal functionality for oblivious transfer that will be used as a building block for the sanitation of randomized equality in Section 4. Secondly, we recap dual-mode cryptosystems and define *sanitizable homomorphic* dual-mode cryptosystems. Finally, we use the latter notion to sanitize the generic framework to instantiate $\mathcal{F}_{\mathsf{OT}}$ of Peikert et al. [25], and extend their concrete instantiation from DDH to verify the newly introduced properties.

3.1 Sanitizable OT

Following the ideas presented in the technical overview in Section 1.3, we report our *sanitized* ideal functionality for oblivious transfer $\mathcal{F}_{\mathsf{sOT}}$ in which both firewalls may intervene in the sanitation of the sender's inputs.

Functionality $\mathcal{F}_{\mathsf{sOT}}$

$\mathcal{F}_{\mathsf{sOT}}$ is a sanitizable ideal functionality that interacts with the sender $\mathrm{S} = (\mathsf{C}_\mathrm{S}, \mathsf{F}_\mathrm{S})$ and the receiver $\mathrm{R} = (\mathsf{C}_\mathrm{R}, \mathsf{F}_\mathrm{R})$, parameterized by input domain $\mathcal{I} \subseteq \{0,1\}^n$ and a blinding operation $* : \mathcal{I}^2 \to \mathcal{I}$.

Interfaces IO_i

Upon receiving a query $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ **from** C_S **on** IO_S**:**

Record $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ and forward the tuple on S_i. Ignore subsequent commands of the form $(\textsc{sender}, \mathsf{sid}, \cdot)$.

Upon receiving a query $(\textsc{receiver}, \mathsf{sid}, \sigma)$ **from** C_R **on** IO_R**:**

Check if a record $(\textsc{sender}, \mathsf{sid}, (\hat{x}_0, \hat{x}_1))$ exists. If this is the case, check the following:

- The message $(\textsc{blind}, \mathsf{sid}, \cdot)$ was sent to $\mathcal{F}_{\mathsf{sOT}}$ on S_S. If P_S is malicious according to the corruption translation table, mark this check as passed.
- The message $(\textsc{blind}, \mathsf{sid}, \cdot)$ was sent to $\mathcal{F}_{\mathsf{sOT}}$ on S_R. If P_R is malicious according to the corruption translation table, mark this check as passed.

If the conditions above hold, output $(\mathsf{sid}, \hat{x}_\sigma)$ to R, sid to the adversary $\mathcal{S}$, and halt. Otherwise, send nothing to R but continue running.

Interfaces S_i

Upon receiving a query $(\textsc{blind}, \mathsf{sid}, (x'_0, x'_1))$ **from** F_S **on** S_S**:**

If P_S is malicious according to the corruption translation table, do nothing. Otherwise, check if a record $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ exists. If so, update the tuple to $(\textsc{sender}, \mathsf{sid}, (\tilde{x}_0, \tilde{x}_1))$, with $\tilde{x}_b = x_b * x'_b$. Otherwise, do nothing. Ignore future commands of the form $(\textsc{blind}, \mathsf{sid}, \cdot)$ on S_S

Upon receiving a query $(\textsc{blind}, \mathsf{sid}, (x''_0, x''_1))$ **from** F_R **on** S_R**:**

If P_R is malicious according to the corruption translation table, do nothing. Otherwise, check the following:

* A record $(\text{SENDER}, \mathsf{sid}, (\tilde{x}_0, \tilde{x}_1))$ exists.
* A message $(\text{BLIND}, \mathsf{sid}, \cdot)$ was sent to $\mathcal{F}_{\mathsf{sOT}}$ on S_S. If P_S is malicious according to the corruption translation table, mark this check as passed.

If the conditions above hold, update the tuple to $(\text{SENDER}, \mathsf{sid}, (\hat{x}_0, \hat{x}_1))$, with $\hat{x}_b = \tilde{x}_b * x''_b$. Otherwise, do nothing. Ignore future commands of the form $(\text{BLIND}, \mathsf{sid}, \cdot)$ on S_R.

We parameterize the blinding operation, yielding a more expressive functionality whose blinding operations may be tailored to the specific input domains of choice (*e.g.*, for additive blinding, $x_0 * x'_0 = x_0 \oplus x'_0$; for multiplicative blinding, $x_0 * x'_0 = x_0 x'_0$).

Furthermore, the functionality disregards blinding inputs sent by corrupted parties (according to the corruption translation table) on behalf of their firewalls. As discussed throughout the technical overview in Section 1.3, this is reasonable, since we are modelling static corruptions and we can represent the components making a corrupted party as a unique party, and we do not want to simulate anything related to messages internally exchanged by the adversary.

3.2 Sanitizable Homomorphic Dual-Mode Encryption

Dual-mode cryptosystems operate like traditional public-key cryptosystems, except for the following differences. First, they introduce the notion of *encryption branches*, in which the key generation algorithm takes as an additional input a branch $\sigma \in \{0,1\}$. The party encrypting the message can choose either branch $b \in \{0,1\}$ over which to encrypt the message. The party receiving the ciphertext is able to decrypt successfully only if $\sigma = b$. Secondly, they rely on a common-reference string that may be setup either in *messy* mode or *decryption* mode. These modes are computationally indistinguishable and induce different algorithms for the generation of a trapdoor, yielding different security guarantees: in messy mode, the sender has statistical security and the receiver has computational security, whereas in decryption mode the security properties are mirrored. We refer to Peikert et al. [25] for further details and report their description of dual-mode cryptosystems in what follows.

Definition 3 (Dual-Mode Cryptosystems)**.** *A dual-mode cryptosystem consists of a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{FindMessy}, \mathsf{TrapKeyGen})$ *with the following properties:*

1. ***Completeness for decryptable branch:*** *For every* $\mu \in \{\mathsf{mes}, \mathsf{dec}\}$*, every* $(\mathsf{crs}, t) \leftarrow_\$ \mathsf{Setup}(1^\lambda, \mu)$*, every* $\sigma \in \{0,1\}$*, every* $(pk, sk) \leftarrow_\$ \mathsf{KeyGen}(\sigma)$*, and every* $m \in \{0,1\}^k$*, decryption is correct on branch* σ*, i.e.,* $\mathsf{Dec}(sk, \mathsf{Enc}(pk, \sigma, m)) = m$*. It also suffices for decryption to be correct with overwhelming probability over the randomness of the entire experiment.*
2. ***Indistinguishability of modes:*** *Let* $\mathsf{SetupMessy}(1^\lambda) = \mathsf{Setup}(1^\lambda, \mathsf{mes})$ *and* $\mathsf{SetupDec}(1^\lambda) = \mathsf{Setup}(1^\lambda, \mathsf{dec})$*. The first outputs of* SetupMessy *and* SetupDec *are computationally indistinguishable, i.e.,* $\mathsf{SetupMessy}_1(1^\lambda) \overset{c}{\approx} \mathsf{SetupDec}_1(1^\lambda)$*.*
3. ***(Messy mode) Trapdoor identification of a messy branch:*** *For every* $(\mathsf{crs}, t) \leftarrow_\$ \mathsf{SetupMessy}(1^\lambda)$ *and every (possibly malformed)* pk*,* $\mathsf{FindMessy}(t, pk)$ *outputs a branch value* $b \in \{0,1\}$ *such that* $\mathsf{Enc}(pk, b, \cdot)$ *is messy. Namely, for every* $m_0, m_1 \in \{0,1\}^k$*,* $\mathsf{Enc}(pk, b, m_0) \overset{s}{\approx} \mathsf{Enc}(pk, b, m_1)$*.*
4. ***(Decryption mode) Trapdoor generation of keys decryptable on both branches:*** *For every* $(\mathsf{crs}, t) \leftarrow_\$ \mathsf{SetupDec}(1^\lambda)$*,* $\mathsf{TrapKeyGen}(t)$ *outputs* (pk, sk_0, sk_1) *such that for every* $\sigma \in \{0,1\}$*,* $(pk, sk_\sigma) \overset{s}{\approx} \mathsf{KeyGen}(\sigma)$*.*

3.2.1 Sanitizable Homomorphic Dual-Mode Cryptosystems.

Looking ahead, we need to augment dual-mode cryptosystems to allow the sanitation of public keys, ciphertexts, and plaintexts related to ciphertexts. We therefore introduce algorithms to enable firewalls to neutralize attacks coming from these attack vectors as follows:

- $\mathsf{HomOp}(c_0, c_1)$, given two ciphertexts c_0, c_1 for messages m_0, m_1 respectively, and an hard-coded operator $*$, outputs a new ciphertext for the message $m_0 * m_1$.
- $(\mathsf{MaulPK}(pk, \rho), \mathsf{AlignEnc}(c, \rho))$. On input a public key pk and a random value ρ, MaulPK produces a new encryption key $\widetilde{pk}$. Any ciphertext $\tilde{c}$ for message m produced under (mauled) encryption key $\widetilde{pk}$ may be transformed with $\mathsf{AlignEnc}(\tilde{c}, \rho)$ into an encryption c of m under (non-mauled) public key pk. Crucially, the randomness has to be the same for consistency to hold.

Intuitively, MaulPK and $\mathsf{AlignEnc}$ are defined as a (symmetric) tuple of algorithms as firewalls will first sanitize the outbound encryption key. Upon receiving messages encrypted under mauled new public key, the firewall will "strip" the layer of sanitation. We are now ready to formally define *sanitizable homomorphic* dual-mode cryptosystems.

Definition 4 (Sanitizable Homomorphic Dual-Mode Cryptosystems)**.** *A sanitizable homomorphic dual-mode cryptosystem consists of a tuple of algorithms* (Setup, KeyGen, Enc, Dec, $\mathsf{FindMessy}$, $\mathsf{TrapKeyGen}$, HomOp, MaulPK, $\mathsf{AlignEnc}$) *with the following properties:*

1. ***Dual-mode cryptosystem:*** *The tuple of algorithms* ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{FindMessy}, \mathsf{TrapKeyGen}$) *constitutes a dual-mode cryptosystem.*
2. ***Homomorphic ciphertexts:*** *For every* $(pk, sk) \leftarrow_{\$} \mathsf{KeyGen}(\sigma)$*, for every* $c_i \leftarrow_{\$} \mathsf{Enc}(pk, \sigma, m_i)$*, with* $i \in \{0,1\}$ *and* $m_i \in \{0,1\}^n$*,* $\mathsf{HomOp}(m_0, m_1)$ *produces a new ciphertext of message* $m_0 * m_1$*, i.e.,* $\mathsf{HomOp}(c_0, c_1) = \mathsf{Enc}(pk, \sigma, (m_0 * m_1))$*.*
3. ***Consistent key sanitation:*** *For every* $\sigma \in \{0,1\}$*, for every* $(pk, sk) \leftarrow_{\$} \mathsf{KeyGen}(\sigma)$*, for every* $\rho \in \{0,1\}^n$*,* $\mathsf{MaulPK}(pk, \rho)$ *outputs a new encryption key* $\widetilde{pk}$ *with the following property. For every* $\tilde{c} \leftarrow_{\$} \mathsf{Enc}(\widetilde{pk}, \sigma, m)$*, with* $i \in \{0,1\}$ *and* $m \in \{0,1\}^n$*,* $\mathsf{AlignEnc}(c, \rho)$ *produces a new ciphertext* c *under public key* pk*, i.e.,* $\mathsf{AlignEnc}(\tilde{c}, \rho) = c$*, where* $c = \mathsf{Enc}(pk, \sigma, m)$*.*

Remark 1. *The* MaulPK*,* $\mathsf{AlignEnc}$*,* HomOp *algorithms are outputting keys and ciphertexts implicitly combining the randomness of their inputs. This property is particularly useful in the context of sanitation, as it allows a firewall to run these algorithms to combine their "good randomness" to destroy subliminal channels stemming from values output by their core with "bad randomness".*

3.3 A Generic Framework

As shown in the generic framework of Peikert *et al.* [25], having access to a dual-mode cryptosystem allows the instantiation of $\mathcal{F}_{\mathsf{OT}}$ in a natural manner: the receiver uses its choice bit σ as the selected decryption branch, and the sender encrypts each of its inputs x_b on a separate encryption branch b. The receiver will be able to decrypt only on branch $\sigma = b$.

3.3.1 Sanitizing the framework.

From a high-level perspective, our sanitized protocol leverages on homomorphic ciphertexts to blind the sender's inputs and using consistent key sanitation to sanitize the receiver's outbound encryption key and realign the inbound ciphertexts for decryption purposes. These operations also destroy any potential subliminal channel linked to the original ciphertexts or to the keys. In Figure 2, we depict a protocol run showing only the firewall of the sender, since the firewall of the receiver behaves exactly the same way.

Theorem 1. *The protocol in Figure 2, parameterized by a mode* $\mathsf{mode} \in \{\mathsf{mes}, \mathsf{dec}\}$*, realizes the sanitizable functionality* $\mathcal{F}_{\mathsf{sOT}}$ *in the* $(\mathcal{F}_{\mathsf{SAT}}, \mathcal{F}_{\mathsf{crs}})$*-hybrid model under static corruptions. For* $\mathsf{mode} = \mathsf{mes}$*, the sender's security is statistical and the receiver's security is computational; for* $\mathsf{mode} = \mathsf{dec}$*, the security properties are reversed.*

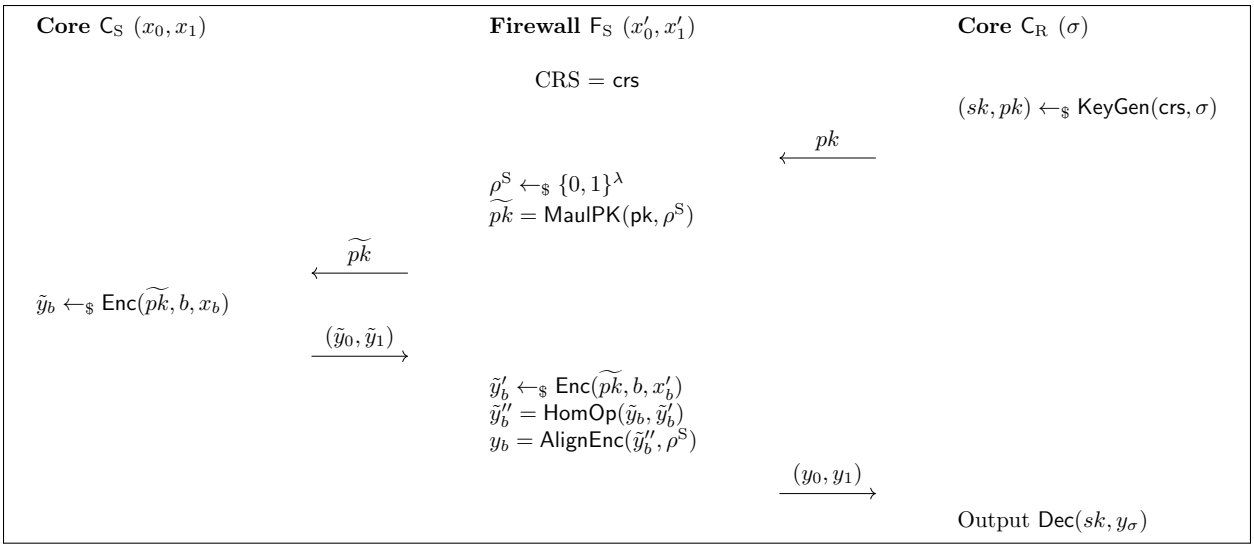

Figure 2: A sanitation of the generic framework of Peikert et al. [25], realizing $\mathcal{F}_{\mathsf{sOT}}$. The receiver's firewall is not explicitly shown, as it runs the same code as F_S.

We first show that the firewalls are *strongly sanitizing*, meaning that the environment is unable to distinguish a protocol run in which either core is speciously corrupted and their firewall is honest from a run in which the same core is incorruptible and their firewall is honest.

Lemma 1. *The firewall* F_S *of the sender* C_S *in Figure 2 is strongly sanitizing.*

Proof (Lemma 1). We show that no environment $\mathcal{E}$, which allows specious corruptions for the core C_S but does not corrupt the respective firewall F_S, is able to distinguish between an execution of the protocol in Figure 2 with an incorruptible C_S from one that accepts the corruption $\tilde{\mathsf{C}}_\mathrm{S}$.

We first argue that no input-trigger attacks are possible. The only message the sender receives from a (potentially malicious) receiver is the public key pk used in the encryption scheme, which may be crafted in a way to trigger a specious corruption in C_S in an undetectable manner. However, F_S randomizes the inbound key by means of algorithm MaulPK, producing a new uniformly distributed key. If the message cannot be parsed as a public key for the encryption scheme, the message is dropped.

We now show that C_S is unable to leak any information. Since the only outbound message of the sender is a tuple of ciphertexts of a dual-mode cryptosystem (y_0, y_1), $\tilde{\mathsf{C}}_\mathrm{S}$ can modify how this tuple is generated. In order to establish a side-channel, the specious core may (i) produce signaling ciphertexts distributed as valid ciphertexts (*e.g.*, by means of rejection sampling), or (ii) produce a different message. $\mathcal{E}$ is unable to determine whether the specious corruption had effect or not, as:

- In case (i) the ciphertext gets replaced by F_i with a fresh ciphertext that is side-channel free, as the firewall the random coin ρ used in $\mathsf{AlignEnc}$ and the internal coins tossed in the computation of (y'_0, y'_1) are sampled uniformly at random.
- In case (ii), if the message cannot be parsed as a tuple of ciphertexts, the message is dropped. Otherwise, the HomOp and $\mathsf{AlignEnc}$ algorithms produce a uniformly distributed value as y_0, y_1 are not valid ciphertexts.

□

Lemma 2. *The firewall* F_R *of the receiver* C_R *in Figure 2 is strongly sanitizing.*

Proof (Lemma 2). We show that no environment $\mathcal{E}$, which allows specious corruptions for the core C_R but does not corrupt the respective firewall F_R, is able to distinguish between an execution of the protocol in Figure 2 with an incorruptible C_R from one that accepts the corruption $\tilde{\mathsf{C}}_\mathrm{R}$.

We first argue that no input-trigger attacks are possible. The only message the receiver receives from a (potentially malicious) sender is the tuple of ciphertexts (y_0, y_1), which may be crafted in a way to trigger a specious corruption in C_R in an undetectable manner. However, F_R randomizes the inbound tuple by means of algorithms HomOp and $\mathsf{AlignEnc}$, producing a new uniformly distributed tuple. If the message cannot be parsed as a tuple of ciphertexts, the message is dropped.

We now show that C_S is unable to leak any information. Since the only outbound message of the receiver is a public key pk of a dual-mode cryptosystem, $\tilde{\mathsf{C}}_\mathrm{R}$ can modify how pk is generated in order to exfiltrate information. The specious core may (i) produce a signaling key distributed as a valid key (*e.g.*, by means of rejection sampling), or (ii) produce a different message. $\mathcal{E}$ is unable to determine whether the specious corruption had effect or not, as:

- In case (i), the key is mauled by F_R to a different key, as the firewall samples the random coin ρ used in MaulPK uniformly at random, and the output key is uniformly distributed.
- In case (ii), if the message cannot be parsed as a public key for the encryption scheme, the message is dropped. Otherwise, the MaulPK algorithm produces a uniformly distributed value, since pk is not a valid key.

□

Lemma 3. *For every adversary $\mathcal{A}$ corrupting either party maliciously and the firewall of the other (honest) party semi-honestly in an execution of the protocol Π as in Figure 2 in the $(\mathcal{F}_{\mathsf{SAT}}, \mathcal{F}_{\mathsf{crs}})$-hybrid model, there exists a simulator $\mathcal{S}$ such that, for all environments $\mathcal{E}$, the following holds:*

$$\textsc{Exec}^{(\mathcal{F}_{\mathsf{SAT}},\mathcal{F}_{\mathsf{crs}})}_{\Pi,\mathcal{A},\mathcal{E}} \approx \textsc{Exec}_{\mathcal{F}_{\mathsf{sOT}},\mathcal{S},\mathcal{E}}$$

Proof (Lemma 3). The proof of this lemma follows closely the proof of the original generic framework of Peikert et al. (Proof of Theorem 4.1, [25]): the dual-mode cryptosystem induces statistical security for the sender in messy mode and statistical security for the receiver in decryption mode. The indistinguishability of modes induces computational security for the specific party in the protocol run.

In the context of our framework, for each corruption scenario, the adversary can semi-honestly corrupt the firewall of the matching honest core. Intuitively, revealing this information does not provide the adversary any advantage, as semi-honestly corrupting the firewall only reveals blinding factors and the matching core is honest.

We analyze corruption scenarios separately, interacting with $\mathcal{F}_{\mathsf{sOT}}$ and the environment $\mathcal{E}$, and simulating a run of the protocol Π for the core of the honest party. We show the proof for $\mathsf{mode} = \mathsf{mes}$; the proof for $\mathsf{mode} = \mathsf{dec}$ only varies in the generation of the common reference strings by the simulator.

Malicious Receiver. The receiver R is corrupted, so the core of the sender C_S is honest and its firewall F_S is semi-honest. $\mathcal{S}$ generates the CRS in messy mode, obtaining a trapdoor t that enables it to find a messy branch related to a (possibly malformed) public key pk. It then initializes the real-world adversary $\mathcal{A}$ with crs.

When $\mathcal{A}$ sends a message containing public key pk, $\mathcal{S}$ extracts a messy branch $b = \mathsf{FindMessy}(\mathsf{crs}, t, pk)$. Then, it sends to $\mathcal{F}_{\mathsf{sOT}}$ the following queries:

1. $(\textsc{blind}, \mathsf{sid}, x'_0, x'_1)$ on S_S.
2. $(\textsc{receiver}, \mathsf{sid}, 1-b)$ on IO_R, receiving $\hat{x}_{1-b} = x_{1-b} * x'_{1-b}$.

Notice that the blind query on S_S is allowed, as the firewall of the sender F_S is semi-honestly corrupted, so its inputs (x'_0, x'_1) are known to $\mathcal{S}$. On the other hand, the query on S_R is not needed, as P_R is corrupted according to the corruption translation table.

Finally, whenever the dummy S is activated for session sid, $\mathcal{S}$ first computes $x_{1-b} = \hat{x}_{1-b} *^{-1} x'_{1-b}$ [3], computes $y_{1-b} \leftarrow_\$ \mathsf{Enc}(pk, 1-b, x_{1-b})$ and $y_b \leftarrow_\$ \mathsf{Enc}(pk, b, 0^n)$, and sends $(\mathsf{sid}, y_{1-b}, y_b)$ to $\mathcal{A}$.

Since the only difference between the real world and ideal world is in the generation of y_b, and b is the messy branch correctly identified by the trapdoor, $\mathsf{Enc}(pk, b, 0^n) \overset{s}{\approx} \mathsf{Enc}(pk, b, x_b)$. The two experiments are therefore statistically indistinguishable.

Malicious Sender. The sender S is corrupted, so the core of the receiver C_R is honest and its firewall F_S is semi-honest. $\mathcal{S}$ generates the CRS in decryption mode, generating a tuple of keys (pk, sk_0, sk_1) with sk_b being the decryption key of the b-th branch for $b \in \{0, 1\}$.

Whenever $\mathcal{A}$ sends $(\mathsf{sid}, \hat{y}_0, \hat{y}_1)$, $\mathcal{S}$ computes $x_b = \mathsf{Dec}(sk_b, \hat{y}_b)$. Then, it sends to $\mathcal{F}_{\mathsf{sOT}}$ the following queries:

1. $(\textsc{Send}, \mathsf{sid}, x_0, x_1)$ on IO_S.
2. $(\textsc{blind}, \mathsf{sid}, x''_0, x''_1)$ on S_R.

Notice that the blind query on S_R is allowed, as the firewall of the receiver F_R is semi-honestly corrupted, so its inputs (x''_0, x''_1) are known to $\mathcal{S}$. On the other hand, the query on S_S is not needed, as P_S is corrupted according to the corruption translation table.

Since the only difference between the real world and ideal world is in the generation of the keys, and $(pk, sk_\sigma) \overset{s}{\approx} \mathsf{KeyGen}(\mathsf{crs}, \sigma)$ by trapdoor key generation. However, crs was generated in decryption mode, whereas the protocol expects messy mode. Since $\mathsf{SetupMessy}_1 \overset{c}{\approx} \mathsf{SetupDec}_1$, the two experiments are only computationally indistinguishable. □

Proof (Theorem 1). The theorem statement follows by looking at the standard corruption translation table. Since the adversary maliciously corrupts either the sender or the receiver, the remaining honest party has either (i) a Honest core and a SemiHonest firewall, or (ii) a Specious core and a Honest firewall. Since both firewalls are strongly sanitizing, as shown in Lemma 1 and Lemma 2, the core in case (ii) can be considered Honest (Lemma 2 of [15]). Hence, the statement follows directly by Lemma 3. We consider Isolate corruptions as Malicious corruptions. □

3.4 Construction from DDH

We briefly recap the instantiation from DDH of Peikert *et al.* [25] of dual-mode cryptosystem (Section 5, [25]). In what follows, we denote $\mathbb{G}$ as the group description on a cyclic group G of prime order p for which DDH is hard, with generators g, h.

- The CRS is a tuple (g_0, h_0, g_1, h_1), with different trapdoors according to the mode of operation.
- $\mathsf{KeyGen}(\sigma) = ((g_\sigma^r, h_\sigma^r), r) = ((pk_1, pk_2), sk) = (pk, sk)$.
- $\mathsf{Enc}(pk, m, b) = (g_b^s h_b^t, pk_1^s pk_2^t m) = (c_1, c_2)$.
- $\mathsf{Dec}(sk, c) = c_2 / c_1^r$.

It turns out that the the DDH cryptosystem is compatible with all the additional interfaces we introduced in Definition 4, and we can define algorithms matching the newly introduced properties in a straight-forward manner:

[3] We assume the operator supports the inverse operation. This is the case for additive blinding ($\ominus$), and multiplicative blinding (multiplicative inverse).

- $\mathsf{MaulPK}(pk, \rho)$: Output pk^ρ.
- $\mathsf{AlignEnc}(c, \rho)$: Parse $c = (c_1, c_2)$. Output $\tilde{c} = (c_1^\rho, c_2)$.
- $\mathsf{HomOp}(c_0, c_1)$: Output $c_0 c_1$.

Theorem 2. *The DDH cryptosystem of Peikert* et al. *[25] with the additional algorithms specified above is a malleable dual-mode cryptosystem, assuming that DDH is hard for* $\mathbb{G}$.

Proof (Theorem 2). Correctness and all the properties related to the original dual-mode cryptosystem are unchanged (Theorem 5.2 [25]). It remains to show that the cryptosystem has homomorphic ciphertexts and consistent key sanitation.

The homomorphic ciphertexts property follows by observing that the underlying DDH cryptosystem is multiplicative homomorphic. In the following, $\mathsf{crs} = (g_0, g_1, h_0, h_1)$, $(pk, sk) = ((g_\sigma^r, h_\sigma^r), r)$, $\widetilde{pk}_{\mathsf{DDH,b}} = (g_b, h_b, \widetilde{pk}_0, \widetilde{pk}_1) = (g_b, h_b, g_\sigma^r, h_\sigma^r)$, and s, s', t, t' are random coins sampled by the DDHEnc algorithm.

$$\begin{aligned}
\mathsf{Enc}(pk, b, m_0)\mathsf{Enc}(pk, b, m_1) &= \mathsf{DDHEnc}(\widetilde{pk}_{\mathsf{DDH,b}}, m_0)\mathsf{DDHEnc}(\widetilde{pk}_{\mathsf{DDH,b}}, m_1) \\
&= (g_b^s h_b^t, g_\sigma^{rs} h_\sigma^{rt} m_0)(g_b^{s'} h_b^{t'}, g_\sigma^{rs'} h_\sigma^{rt'} m_1) \\
&= (g_b^{(s+s')} h_b^{(t+t')}, g_\sigma^{r(s+s')} h_\sigma^{r(t+t')} m_0 m_1) \\
&= \mathsf{DDHEnc}(\widetilde{pk}_{\mathsf{DDH,b}}, m_0 m_1) \\
&= \mathsf{Enc}(pk, b, m_0 m_1)
\end{aligned}$$

The consistent key sanitation property follows by inspection. In the following, $\mathsf{crs} = (g_0, g_1, h_0, h_1)$, $(pk, sk) = ((g_\sigma^r, h_\sigma^r), r)$, $\widetilde{pk} = \mathsf{MaulPK}(pk, \rho) = (g_\sigma^{r\rho}, h_\sigma^{r\rho})$, and $c \leftarrow_\$ \mathsf{Enc}(\widetilde{pk}, \sigma, m)$, and s, t are random coins sampled by the DDHEnc algorithm.

$$\begin{aligned}
\mathsf{Dec}(sk, \mathsf{AlignEnc}(c, \rho)) &= \mathsf{Dec}(sk, (c_1^\rho, c_2)) \\
&= c_2/(c_1^\rho)^r \\
&= g_\sigma^{r\rho s} h_\sigma^{r\rho t} m / g_\sigma^{s\rho r} h_\sigma^{t\rho r} \\
&= m
\end{aligned}$$

□

4 Sanitizing Randomized Equality

In this section, we present our sanitized protocol for the (regular) randomized equality ideal functionality $\mathcal{F}_{\mathsf{RE}}$ that relies on authenticated channels (i.e., $\mathcal{F}_{\mathsf{SAT}}$) and $\mathcal{F}_{\mathsf{sOT}}$, following the construction of Canetti *et al.* [10].

4.1 Randomized Equality and PAKE

We describe a variation of the randomized equality ideal functionality $\mathcal{F}_{\mathsf{RE}}$ of Barak *et al.* [5] which we will realize, and point out its differences with respect to PAKE. Crucially, we discuss both RE and PAKE without explicit mutual authentication. As discussed in the technical overview in Section 1.3, we stress that this property is not as trivial to achieve as in MPC without subversion.

4.1.1 Randomized Equality.

$\mathcal{F}_{\mathsf{RE}}$ features an initiator I and a responder R who obtain the same random high-entropy value skey only if they have input the same passwords. More specifically, the initiator declares its intention to run the functionality

with the responder by means of an INIT query with password w^{I}. The adversary $\mathcal{S}$ is notified of this interaction, and controls the delivery time of the initialization message destined to the receiver. Whenever this occurs, the responder inputs its own password w^{R} with a RESPOND query, and the functionality prepares the output for both parties depending on whether $w^{\mathrm{R}} = w^{\mathrm{I}}$ or not. In the former case, the output will be the same random key skey for both parties; in the latter, two keys sampled independently at random. Finally, the adversary controls the delivery of the output of parties by means of an OUT query, in which it also specifies a key K. If either party involved in the interaction is corrupted and the passwords match, both parties will receive the adversarially-generated key K. Otherwise, they will receive whatever has been stored in $\mathsf{out}_{\mathsf{P}}$.

Functionality $\mathcal{F}_{\mathsf{RE}}$

The functionality $\mathcal{F}_{\mathsf{RE}}$ is parameterized by a security parameter λ and a dictionary $\mathcal{D}$. It interacts with an initiator I, a responder R, and the adversary $\mathcal{S}$ via the following messages:

Upon receiving a query $(\text{INIT}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, w^{\mathrm{I}})$, $w^{\mathrm{I}} \in \mathcal{D}$ **from** I**:**

Record $(\mathrm{I}, \mathrm{R}, w^{\mathrm{I}})$ and send a message $(\mathsf{sid}, \mathrm{I}, \mathrm{R})$ to $\mathcal{S}$. Ignore all future messages from I.

Upon receiving a query $(\text{OK}, \mathsf{sid})$ **from** $\mathcal{S}$**:**

Send a message $(\text{WAKEUP}, \mathsf{sid}, \mathrm{I}, \mathrm{R})$ to R. Ignore all future (OK) messages.

Upon receiving a query $(\text{RESPOND}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, w^{\mathrm{R}})$ **from** R**:**

- If $w^{\mathrm{R}} = w^{\mathrm{I}}$, choose $\mathsf{skey} \leftarrow_{\$} \{0,1\}^{\lambda}$ and store $\mathsf{out}_{\mathrm{I}} = \mathsf{out}_{\mathrm{R}} = \mathsf{skey}$.
- If $w^{\mathrm{R}} \neq w^{\mathrm{I}}$, then set $\mathsf{out}_{\mathrm{I}} \leftarrow_{\$} \{0,1\}^{\lambda}$, $\mathsf{out}_{\mathrm{R}} \leftarrow_{\$} \{0,1\}^{\lambda}$.

In both cases, ignore subsequent inputs from R.

Upon receiving a query $(\text{OUT}, \mathsf{sid}, \mathsf{P}, K)$, $\mathsf{P} \in \{\mathrm{I}, \mathrm{R}\}$ **from** $\mathcal{S}$**:**

- If $w^{\mathrm{I}} = w^{\mathrm{R}}$ and either I or R are corrupted, or P is corrupted, output (sid, K) to party P.
- Otherwise, output $(\mathsf{sid}, \mathsf{out}_{\mathsf{P}})$ to party P.

Ignore all subsequent (OUT, P) queries for the same party P.

4.1.2 Password-Authenticated Key Exchange

The interaction is conceptually close to that of $\mathcal{F}_{\mathsf{RE}}$, with the important difference that the adversary is now allowed to perform (online) password guesses. In $\mathcal{F}_{\mathsf{RE}}$, the output of parties was determined as soon as the responder had input its password, with the possibility for the adversary to force the output of its own adversarially-generated key K only by actively controlling a party and knowing the password of its peer. Here, the adversary can also attempt to guess the password of either party by means of TESTPWD queries to the functionality without even corrupting them!

Compared to $\mathcal{F}_{\mathsf{RE}}$, each record now has a status associated with it. If the adversary performs a password guess against party P and fails, the target record is marked as $\mathsf{interrupted}$. If instead the password guess succeeds, the target record is marked as $\mathsf{compromised}$, and P will output the adversarially-generated key K. The adversary can now disclose the output of the functionality first to the initiator and then to the responder: the same (random) key is returned to both parties only if they have the same password and the following conditions hold: (i) the initiator's record is fresh and is queried first, and (ii) the responder's record is fresh. In any other case, the functionality outputs random keys.

Functionality $\mathcal{F}_{\mathsf{PAKE}}$

The functionality $\mathcal{F}_{\mathsf{PAKE}}$ is parameterized by a security parameter λ. an initiator I, a responder R, and the adversary $\mathcal{S}$ via the following queries:

Upon receiving a query $(\textsc{NewSession}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, w^{\mathrm{I}})$ **from** I**:**

Record $(\mathrm{I}, \mathrm{R}, w^{\mathrm{I}})$, mark it as fresh, and leak $(\mathsf{sid}, \mathrm{I}, \mathrm{R})$ to $\mathcal{S}$. Ignore all future messages from I.

Upon receiving a query $(\textsc{ok}, \mathsf{sid})$ **from** $\mathcal{S}$**:**

Send a message $(\textsc{NewSession}, \mathsf{sid}, \mathrm{I}, \mathrm{R})$ to R. Ignore all future $(\textsc{ok})$ messages.

Upon receiving a query $(\textsc{Respond}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, w^{\mathrm{R}})$ **from** R**:**

Record $(\mathrm{R}, \mathrm{I}, w^{\mathrm{R}})$ and mark it as fresh.

Upon receiving a query $(\textsc{TestPwd}, \mathsf{sid}, \mathsf{P}, w')$ **from the adversary** $\mathcal{S}$**:**

If $\mathsf{P} \in \{\mathrm{I}, \mathrm{R}\}$ and there is a record of the form $(\mathsf{P}, \cdot, w)$ which is fresh, then:

- If $w' = w$, mark the record as $\mathsf{compromised}$ and return "correct guess" to $\mathcal{S}$.
- If $w' \neq w$, mark the record as $\mathsf{interrupted}$ and return "wrong guess" to $\mathcal{S}$.

Upon receiving a query $(\textsc{NewKey}, \mathsf{sid}, \mathsf{P}, K)$ **from** $\mathcal{S}$**, where** $|K| = \lambda$**:**

If $\mathsf{P} \in \{\mathrm{I}, \mathrm{R}\}$ and there is a record of the form $(\mathsf{P}, \cdot, w)$ that is not marked as $\mathsf{completed}$ then:

- If this record is $\mathsf{compromised}$, or either $w^{\mathrm{I}} = w^{\mathrm{R}}$ and I or R is corrupted, or P is corrupted, then output (sid, K) to party P.
- If this record is fresh, $\mathsf{P} = \mathrm{I}$, and there exists a record $(\mathrm{R}, \mathrm{I}, w^{\mathrm{R}})$ with $w^{\mathrm{R}} = w$, choose $\mathsf{skey} \leftarrow_{\$} \{0,1\}^{\lambda}$. Output skey to I, and append skey to the record $(\mathrm{I}, \mathrm{R}, w^{\mathrm{I}})$.
- If this record is fresh, $\mathsf{P} = \mathrm{R}$, and there exists a record $(\mathrm{I}, \mathrm{R}, w^{\mathrm{I}}, \mathsf{skey})$ with $w^{\mathrm{I}} = w$, output skey to R.
- If none of the above rules apply, choose $\mathsf{skey}' \leftarrow_{\$} \{0,1\}^{\lambda}$ and output it to party P.

In any case, mark the record $(\mathsf{P}, \cdot, w)$ as $\mathsf{completed}$.

4.2 Randomized Equality from OT

We sanitize the RE from OT protocol of Canetti *et al.* [10] by using $\mathcal{F}_{\mathsf{sOT}}$. Compared to the non-sanitized protocol, we parameterize the input domain $\mathcal{I}$ and the respective blinding operation $*$, in line with the description of $\mathcal{F}_{\mathsf{sOT}}$. For the ease of exposition, we depict the protocol in Figure 3 assuming 1-bit passwords. The n-bit password case runs exactly in the same way except that (i) it requires n OTs, and (ii) it computes keys using operator $*$ with n random coins rather than only one.

In order to preserve correctness, we leverage on the symmetry of the protocol. In particular, the values each party retrieves from the batch of OTs in which they act as receivers embeds the random strings that are used by both firewalls, and these strings are the same also for the other OT batch. This also thwarts both input triggering attacks, as well as information leakage.

Theorem 3. *The protocol in Figure 3 wsrUC-realizes the $\mathcal{F}_{\mathsf{RE}}$ ideal functionality in the $(\mathcal{F}_{\mathsf{sOT}}, \mathcal{F}_{\mathsf{SAT}})$-hybrid model under static corruptions.*

First of all, we argue that the protocol is correct.

Lemma 4. *The protocol in Figure 3 is correct.*

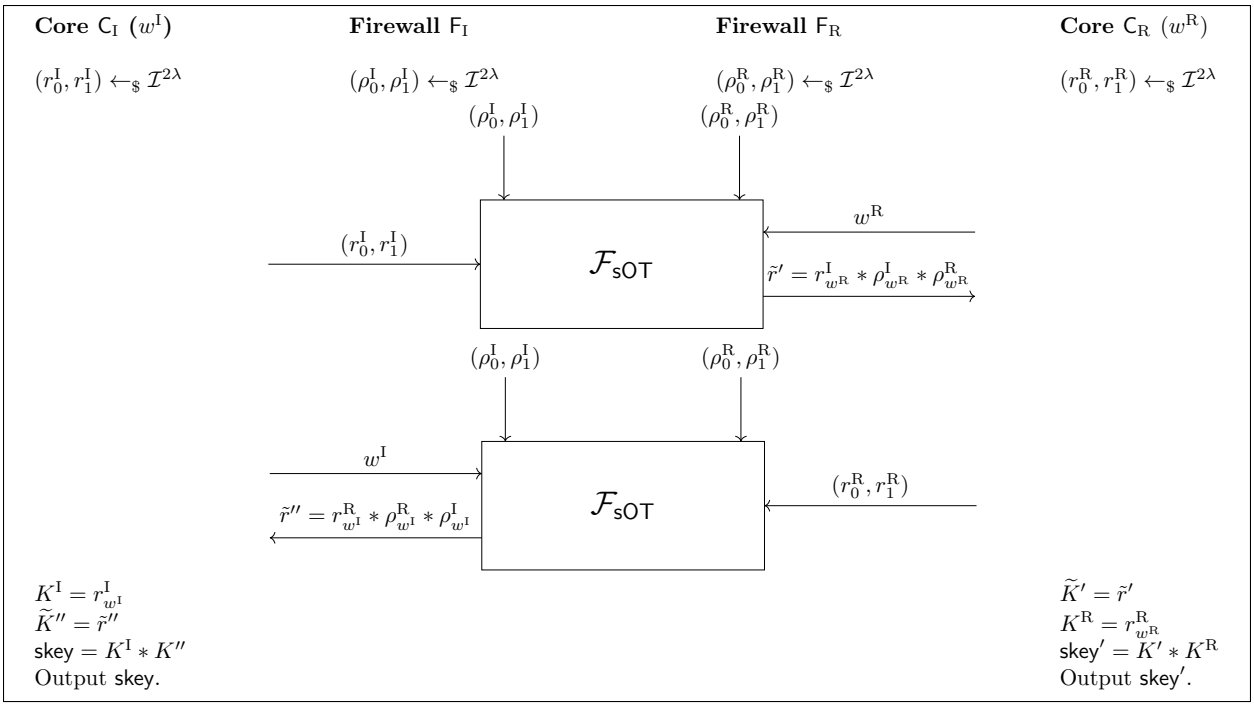

Figure 3: A sanitizing protocol for $\mathcal{F}_{\mathsf{RE}}$ from sanitizable OT with a 1-bit password.

Proof (Lemma 4). If $w^\mathrm{I} = w^\mathrm{R}$, then $K^\mathrm{I} * \widetilde{K}'' = \widetilde{K}' * K^\mathrm{R}$ (i.e., parties output the same key):

$$\begin{aligned} K^\mathrm{I} * \widetilde{K}'' &= (r^\mathrm{I}_{w^\mathrm{I}}) * (r^\mathrm{R}_{w^\mathrm{I}} * \rho^\mathrm{R}_{w^\mathrm{I}} * \rho^\mathrm{I}_{w^\mathrm{I}}) \\ &= (r^\mathrm{I}_{w^\mathrm{I}} * \rho^\mathrm{I}_{w^\mathrm{I}} * \rho^\mathrm{R}_{w^\mathrm{I}}) * r^\mathrm{R}_{w^\mathrm{I}} \\ &= (r^\mathrm{I}_{w^\mathrm{R}} * \rho^\mathrm{I}_{w^\mathrm{R}} * \rho^\mathrm{R}_{w^\mathrm{R}}) * r^\mathrm{R}_{w^\mathrm{R}} \\ &= \widetilde{K}' * K^\mathrm{R} \end{aligned}$$

□

We now show that the firewalls are *strongly sanitizing*, meaning that the environment is unable to distinguish a protocol run in which either core is speciously corrupted and their firewall is honest from a run in which the same core is incorruptible and their firewall is honest.

Lemma 5. *The firewall F_P in Figure 3, for $\mathsf{P} \in \{\mathrm{I}, \mathrm{R}\}$, is strongly sanitizing.*

Proof (Lemma 5). We show that no environment $\mathcal{E}$, which allows specious corruptions for the core C_P but does not corrupt the respective firewall F_P, is able to distinguish between an execution of the protocol in Figure 3 with an incorruptible C_P from one that accepts the corruption $\tilde{\mathsf{C}}_\mathsf{P}$. We consider the case $\mathsf{P} = \mathrm{I}$. Since the protocol is symmetric, the proof is the same for both parties.

We first argue that no input-trigger attacks are possible. The only source of input-triggers comes from the inbound values of the second batch of $\mathcal{F}_{\mathsf{sOT}}$, as the inputs of the (malicious) responder may have been crafted in a way to trigger a specious corruption in C_I in an undetectable manner. Since F_I samples coins ρ_b^I uniformly at random from the input space $\mathcal{I}$, the output of the operation $*$ is uniformly random in $\mathcal{I}$, even in the presence of a biased distribution sampled by a malicious responder.

We now show that C_I is unable to leak any information. The initiator's outbound communication consists of choosing random coins and sending them through $\mathcal{F}_{\mathsf{sOT}}$. Since F_I samples coins ρ_b^I uniformly at random

from the input space $\mathcal{I}$, the output of the operation $*$ is uniformly random in $\mathcal{I}$, even in the presence of a biased distribution sampled by the specious core $\tilde{\mathsf{C}}_{\mathrm{I}}$. □

Lemma 6. *For every adversary $\mathcal{A}$ corrupting either party maliciously and the firewall of the other (honest) party semi-honestly in an execution of the protocol Π as in Figure 2 in the $(\mathcal{F}_{\mathsf{sOT}}, \mathcal{F}_{\mathsf{SAT}})$-hybrid model, there exists a simulator $\mathcal{S}$ such that, for all environments $\mathcal{E}$, the following holds:*

$$\textsc{Exec}^{(\mathcal{F}_{\mathsf{sOT}},\mathcal{F}_{\mathsf{SAT}})}_{\Pi,\mathcal{A},\mathcal{E}} \approx \textsc{Exec}_{\mathsf{Wrap}(\mathcal{F}_{\mathsf{RE}}),\mathcal{S},\mathcal{E}}$$

Proof (Lemma 6). The proof of this lemma follows closely the proof of the non-sanitized protocol of Canetti et al. (Proof of Theorem 1, [25]), restricted to the case of static corruptions.

We analyze corruption scenarios separately, interacting with $\mathcal{F}_{\mathsf{RE}}$ and the environment $\mathcal{E}$, and simulating a run of the protocol Π for the core of the honest party.

Malicious Initiator. The initiator I is corrupted, so the core of the receiver C_{R} is honest and its firewall F_{R} is semi-honest. The simulator $\mathcal{S}$ can simulate the ideal functionality $\mathcal{F}_{\mathsf{sOT}}$, hence it can obtain whatever the adversary $\mathcal{A}$ inputs on behalf of corrupted parties (honest parties are still dummy parties as per the UC framework). More specifically, it simulates the first batch of OTs (in which I acts as the sender) as follows:

1. Whenever $\mathcal{A}$ sends inputs $(r^{\mathrm{I}}_0, r^{\mathrm{I}}_1)$ to $\mathcal{F}_{\mathsf{sOT}}$ acting as a malicious initiator in the first OT, $\mathcal{S}$ records the tuple.
2. Whenever $\mathcal{A}$ sends inputs $(\rho^{\mathrm{R}}_0, \rho^{\mathrm{R}}_1)$ to $\mathcal{F}_{\mathsf{sOT}}$ acting as the receiver's firewall F_{R} in the first OT, $\mathcal{S}$ updates the tuple as the functionality would, provided it has received a message as in (1). Otherwise, it does nothing.

Notice that step (1) is still necessary, as the functionality still expects the semi-honest F_{R} to blind the sender's inputs of the OT (as the overall party is honest). It then simulates the second batch of OTs (in which I acts as the receiver) as follows:

1. Whenever $\mathcal{A}$ sends inputs $(\rho^{\mathrm{R}}_0, \rho^{\mathrm{R}}_1)$ to $\mathcal{F}_{\mathsf{sOT}}$ acting as the responder's firewall F_{R} in the second OT, $\mathcal{S}$ locally tracks that it has received such a query but drops the actual values.
2. Whenever $\mathcal{A}$ sends input w to $\mathcal{F}_{\mathsf{sOT}}$ acting as a malicious initiator in the second OT, if $\mathcal{S}$ had received a message as in (1), it records the input w, and returns a random value $r' \leftarrow_{\$} \mathcal{I}$ to $\mathcal{A}$. Otherwise, it does nothing.

This is allowed, as r' is a uniformly random value generated by $\mathcal{S}$ independently from ρ^{R}_w (and this would be the expected behaviour of the C_{R}).

After this preliminary phase, $\mathcal{S}$ sends message $(\textsc{Init}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, w)$ to $\mathcal{F}_{\mathsf{RE}}$. It also sends message $(\textsc{ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{RE}}$, prompting the (dummy) receiver to input its password. Then, it computes the key as the initiator would in the protocol by using the password w it extracted from the adversary. More specifically, $K^{\mathrm{I}} = r^{\mathrm{I}}_w$, and $\widetilde{K}'' = r'$. $\mathcal{S}$ then sends an $\textsc{Out}$ query containing $K = K^{\mathrm{I}} * \widetilde{K}''$ to $\mathcal{F}_{\mathsf{RE}}$ for both parties, receiving $\mathsf{out}_{\mathrm{I}}$ (i.e., the output of the corrupted initiator).

Malicious Receiver. $\mathcal{S}$ initially waits for a message $(\mathsf{sid}, \mathrm{I}, \mathrm{R})$ from $\mathcal{F}_{\mathsf{RE}}$. As soon as the message is received, it inputs $(\textsc{ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{RE}}$, receiving a matching $(\textsc{wakeup}, \mathsf{sid}, \mathrm{I}, \mathrm{R})$ message. From this point on, it simulates the $\textsc{Respond}$ query exactly as the simulation for the malicious initiator. □

Proof (Theorem 3). The theorem statement follows by looking at the standard corruption translation table. Since the adversary maliciously corrupts either the initiator or the responder, the remaining honest party has either (i) a $\textsc{Honest}$ core and a $\textsc{SemiHonest}$ firewall, or (ii) a $\textsc{Specious}$ core and a $\textsc{Honest}$ firewall. Since both firewalls are strongly sanitizing, as shown in Lemma 5, the core in case (ii) can be considered $\textsc{Honest}$ (Lemma 2 of [15]). Hence, the statement follows directly by Lemma 6. We consider $\textsc{Isolate}$ corruptions as $\textsc{Malicious}$ corruptions. □

5 Subversion-Resilient Split Functionalities

In this section, we extend the notion of split functionalities of Barak *et al.* [5] to the srUC framework. Informally, we want to show that any protocol π wsrUC-realizing a 2-party functionality $\mathcal{F}$ assuming authenticated channels can be compiled into another protocol Π that wsrUC-realizes the 2-party split functionality $s\mathcal{F}$ in the *unauthenticated* channel setting.[4] More formally, the goal of this section is to prove the following theorem.

Theorem 4. *Let $\mathcal{F}$ be a (regular) 2-party UC functionality that can be realized in the wsrUC model given authenticated communication (i.e., $\mathcal{F}_{\mathsf{SAT}}$). Then, assuming the existence of sanitizable signatures, there exists a protocol Π_F that securely realizes the 2-party split functionality $s\mathcal{F}$ in the wsrUC model.*

Towards that, we follow the same strategy as [5] and proceed in the following three stages:

- *Link initialization*: The first step consists in building the *sanitizable* split-authenticated functionality $s\mathcal{F}_{\mathsf{SAT}}$ that parties will use to communicate on. The $s\mathcal{F}_{\mathsf{SAT}}$ functionality can be seen as the split version of the $\mathcal{F}_{\mathsf{SAT}}$ functionality.
- *Multi-session security*: As the second step, we show that when authenticated channels are available, any functionality can be "poly-realized" in the wsrUC model. Here, poly-realizing a functionality informally means that security of the protocol implementing the functionality still holds even when multiple (*i.e.*, poly-many) instances of the protocol share the *same setup*. For that, we show that the subversion-resilient GMW protocol from [15] poly-realizes any functionality in the wsrUC model.
- *Unauthenticated channels*: Finally, we show that the generic transformation of [5] from any protocol π that 2-realizes a 2-party functionality $\mathcal{F}$ given authenticated channels in the srUC model into a protocol Π that realizes $s\mathcal{F}$ given access to $s\mathcal{F}_{\mathsf{SAT}}$ in the srUC model. This is achieved by adapting the transformation of [5] to the srUC model.

Next, we look at each of these stages individually towards demonstrating Theorem 4.

5.1 Building Link Initialization

In this section we formally define $s\mathcal{F}_{\mathsf{SAT}}$ (*i.e.*, the split version of the sanitizable authenticated channel functionality $\mathcal{F}_{\mathsf{SAT}}$ of [15]) and we build a protocol that realizes it in the 2-party setting in the srUC model. For that, we introduce the notion of sanitizable signatures and show that it can be instantiated with the BLS signature scheme [8].

5.1.1 Description of $s\mathcal{F}_{\mathsf{SAT}}$.

The $s\mathcal{F}_{\mathsf{SAT}}$ functionality only guarantees that each party will be interacting with the same entity throughout the entire protocol run. The party could either be communicating with the expected party or with the adversary itself. The $s\mathcal{F}_{\mathsf{SAT}}$ functionality has a similar structure to $\mathcal{F}_{\mathsf{SAT}}$, with the addition of having a link initialization phase. We describe $s\mathcal{F}_{\mathsf{SAT}}$ next.

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

$s\mathcal{F}_{\mathsf{SAT}}$ is a sanitizable ideal functionality that interacts with a set of parties $\mathcal{P}$, each composed of a core C_i and a firewall F_i, and an adversary $\mathcal{S}$. The functionality consists of the following communication interfaces for the cores and the firewalls respectively.

Initialization

[4] Note that we restrict our theorem to 2-party functionalities (in contrast to [5]) as the theorem relies on the sanitizable $s\mathcal{F}_{\mathsf{SAT}}$ functionality that we will only show how to realize for the 2-party setting.

- Upon activation with input $(\textsc{Init}, \mathsf{sid})$, parse $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$ where $\mathcal{P}$ is a set of parties, record $(\mathcal{P}, \mathsf{sid}')$ and forward it to the adversary $\mathcal{S}$.
- Upon receiving the message $(\textsc{Init}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$, from $\mathcal{S}$: Verify that party $\mathsf{P} = (\mathsf{C}, \mathsf{F}) \in \mathcal{P}$, that the list H of party identities includes P, and that for all recorded sets H', either $H \cap H'$ contains only corrupted parties (as defined in the standard corruption transition table) and $\mathsf{sid}_H \neq \mathsf{sid}_{H'}$, or $H' = H$ and $\mathsf{sid}_H = \mathsf{sid}_{H'}$. If so, output $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ to P and record (H, sid_H) if not yet recorded. Else, do nothing.

Message Authentication

- Upon receiving the message $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_j, m)$ on IO_i where $\mathsf{P}_i \in \mathcal{P}$, output the tuple on S_i.
- Upon receiving the message $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_j, \widetilde{m})$ on S_i, add the tuple to an (initially empty) list $\mathcal{W}$ of waiting messages. The same triple can appear multiple times in the list. Then, leak the tuple to $\mathcal{S}$.
- Upon receiving the message $(\textsc{Deliver}, (\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \widetilde{m}))$ from $\mathcal{S}$:
  - If P_j did not previously receive an $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ output then do nothing.
  - Else, if P_i is in the authentication set H of P_j, and P_i is uncorrupted, then: if there is a tuple $(\mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m}) \in \mathcal{W}$, remove one appearance of the triple from $\mathcal{W}$ and output $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \widetilde{m})$ on S_j. Otherwise do nothing.
  - Else (i.e., P_j received $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$, and either P_i is corrupted or $\mathsf{P}_i \notin H$), output $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \widetilde{m})$ on S_j, regardless of $\mathcal{W}$.
- Upon receiving the message $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \hat{m})$ on S_j, output the tuple on IO_j.

The functionality is divided into a preliminary initialization phase and the actual message authentication phase. In the initialization phase, the adversary controls how parties will be partitioned in the respective authentication sets. Intuitively, parties within the same authentication set will be able to communicate as if there was an authenticated channel between them. It is however possible for the adversary to participate in different authentication sets on behalf of all corrupted parties, and any party outside of that authentication set. In the message authentication phase, honest parties will transmit messages in an authenticated fashion within the same authentication set. However, they may very well receive messages out of the blue from the adversary on behalf of any party that is corrupted or outside the authentication set.

With respect to sanitation, whenever a core sends a message m with destination P_j on IO_i, the message is output on S_i. This means that m is output to a firewall that will decide if/how to sanitize m to $\widetilde{m}$ in any arbitrary way, without involving the functionality in the sanitation process. Once the firewall determines the message $\widetilde{m}$ to send to P_j, $\widetilde{m}$ is leaked to the adversary. According to the partition of parties performed in the link authentication phase, the adversary has different capabilities:

- If the recipient party is within the same authentication set, the adversary can exclusively control its delivery time. This behaviour is indeed equivalent to $\mathcal{F}_{\mathsf{SAT}}$, in which the message is stored and then output to the recipient party whenever the adversary decides to do so.
- If P_i is corrupted or the parties are in different authentication sets, the adversary may deliver arbitrary messages to P_j, disregarding the message queue.

Whenever the adversary allows the delivery of a message, that message is output to the firewall F_j. Similarly to the sending phase, F_j may now modify the message arbitrarily without involving the functionality. Once a (potentially different) message $\hat{m}$ is determined by F_j, it is delivered by the functionality to C_j.

We stress that, as it was the case for $\mathcal{F}_{\mathsf{SAT}}$, cores and their respective firewall are allowed to freely communicate through secure channels. This is achieved by means of Send messages (from a core to its

firewall), and RECEIVE messages (from a firewall to its core). In principle, a firewall may send back any message to its core, even if it was not related to any DELIVER message of the adversary.

5.1.2 Comparing channel assumptions.

In Figure 4, we compare the backbone of communication offered by $\mathcal{F}_{\mathsf{SAT}}$ and $s\mathcal{F}_{\mathsf{SAT}}$. We remark that having secure channels between cores and their respective firewall is still required, as otherwise the adversary could simply speciously corrupt a core C_i and analyze the traffic from C_i to F_i, nullifying any sanitation the firewall may have introduced. From a practical standpoint, this assumption is perfectly reasonable. In close-range environments, such as LANs in enterprises or home networks, a user may have its machine connected to a firewall (which could very well be the router itself) by means of a physical cable, and it is reasonable to assume that no wiretapping will occur.

The framework would still hold even in the presence of remote firewalls: a core could establish a secure channel with its firewall via a maliciously-secure 2PC protocol (*e.g.*, by running a maliciously secure 2PC-PAKE) in a preliminary setup phase. Notice that this still makes sense in our threat model, as the adversary would have to mount an attack against a honest party, which is constituted of either a specious core and an honest firewall or an honest core and a semi-honest firewall. In either case, one of the two components remains honest throughout the setup of the secure channel.

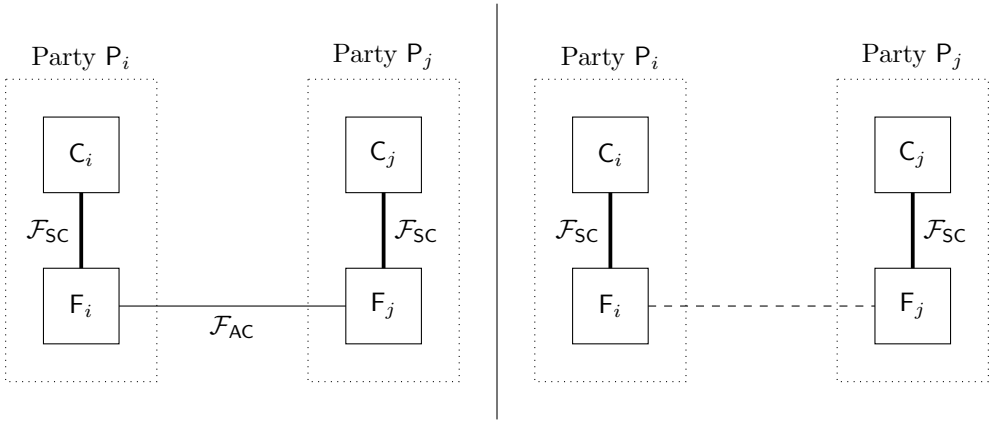

Figure 4: On the left, communication as in $\mathcal{F}_{\mathsf{SAT}}$. On the right, communication as in $s\mathcal{F}_{\mathsf{SAT}}$. $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{F}_{\mathsf{AC}}$ represent secure channels and authenticated channels, respectively.

5.1.3 Sanitizable Signature Schemes.

In the construction of $\mathcal{F}_{\mathsf{SA}}$ of Barak *et al.* [5], parties exchange locally-generated keys and sign their messages in order to preserve the split-authenticated security of the communication channel. However, in order to avoid subversion attacks, both inbound and outbound verification keys have to be appropriately sanitized by firewalls, breaking correctness in the verification of the signature. In order to overcome this limitation, we introduce a new notion that we call *sanitizable* signature schemes.

Informally, a *sanitizable* signature scheme allows to maul the verification key from vk to $\widetilde{\mathsf{vk}}$ by means of an algorithm MaulVK that takes as input randomness ρ. The same randomness may be re-used by an algorithm $\mathsf{AlignSig}$ to align an (accepting) signature σ produced under key sk, producing a signature $\widetilde{\sigma}$ that verifies with mauled key $\widetilde{\mathsf{vk}}$. The latter operation should also be invertible, meaning that the signature σ may be re-computed from $\widetilde{\sigma}$ and ρ. We formally define this notion as a natural extension of EUF-CMA signatures in Definitions 5 and 6, and show that EUF-CMA implies the new security notion in a black-box manner.

Definition 5 (Sanitizable signature schemes)**.** *A sanitizable existentially unforgeable under chosen message attack signature scheme consists of a tuple of algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{MaulVK}, \mathsf{AlignSig}, \mathsf{UnAlignSig})$ *with the following properties:*

1. ***Correctness:*** *For every* $(\mathsf{vk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KeyGen}(1^\lambda)$*, for every* $\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{sk}, m)$ *with* $m \in \{0,1\}^n$*,* $\mathsf{Vrfy}(\mathsf{vk}, (m, \sigma)) = 1$*.*
2. ***Consistent key sanitation:*** *For every* $(\mathsf{vk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KeyGen}(1^\lambda)$*, for every* $\rho \in \{0,1\}^n$*,* $\mathsf{MaulVK}(\mathsf{vk}, \rho)$ *outputs a new verification key* $\widetilde{\mathsf{vk}}$ *with the following property. For every* $\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{sk}, m)$ *with* $m \in \{0,1\}^n$*,* $\mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho)$ *produces an accepting signature* $\widetilde{\sigma}$ *for message* m *verifiable by verification key* $\widetilde{\mathsf{vk}}$*, i.e.,* $\mathsf{Vrfy}(\widetilde{\mathsf{vk}}, \mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho)) = 1$*, where* $\widetilde{\mathsf{vk}} = \mathsf{MaulVK}(\mathsf{vk}, \rho)$ *and* $\sigma = \mathsf{Sign}(\mathsf{sk}, m)$*.*
3. ***Alignment invertibility:*** *For every* $(\mathsf{vk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KeyGen}(1^\lambda)$*, for every* $\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{sk}, m)$ *with* $m \in \{0,1\}^n$*, for every* $\rho \in \{0,1\}^n$*, for every* $\widetilde{\mathsf{vk}} = \mathsf{MaulVK}(\mathsf{vk}, \rho)$*, for every* $\widetilde{\sigma} = \mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho)$*, the algorithm* $\mathsf{UnAlignSig}$ *returns the original signature* σ*, i.e.,* $\sigma = \mathsf{UnAlignSig}((\widetilde{\mathsf{vk}}, \widetilde{\sigma}, m), \rho)$

Definition 6 (Sanitizable EUF-CMA security)**.** *A sanitizable signature scheme is sanitizable existentially unforgeable under chosen message (SAT EUF-CMA) if the probability of the adversary* $\mathcal{A}$ *winning the following game is negligible:*

- *Sample* $(\mathsf{vk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KeyGen}(1^\lambda)$ *and a blinding factor* $\rho \leftarrow_{\$} \{0,1\}^n$*, and run* $\mathcal{A}(\mathsf{vk}, \rho)$*. Compute* $\widetilde{\mathsf{vk}} = \mathsf{MaulVK}(\mathsf{vk}, \rho)$*.*
- *Upon receiving a query from* $\mathcal{A}$ *with message* m*, compute* $\sigma = \mathsf{Sign}(\mathsf{sk}, m)$ *and* $\mathsf{AlignSig}((\mathsf{vk}, m, \sigma), \rho)$*. Respond with* $\widetilde{\sigma}$ *and add* m *to a list* $\mathcal{M}$*.*
- *Challenge* $\mathcal{A}$ *to produce a signature* $\widetilde{\sigma}^*$ *on message* $m^* \notin \mathcal{M}$ *that verifies under* $\widetilde{\mathsf{vk}}$*.*
- *Upon receiving a response* $(m^*, \widetilde{\sigma}^*)$*,* $\mathcal{A}$ *wins if* $\mathsf{Vrfy}_{\widetilde{\mathsf{vk}}}(m^*, \widetilde{\sigma}) = 1$*.*

Theorem 5. *Any EUF-CMA signature scheme is also SAT EUF-CMA.*

Proof. We build a reduction in Figure 5. □

5.1.4 Combining verification keys.

Looking ahead, the link initialization phase of the protocol realizing $s\mathcal{F}_{\mathsf{SAT}}$ relies on the determination of session IDs via (identifying) verification keys of parties, which get sanitized by firewalls in different directions. More specifically, core C_i has access to vk_i and $\widetilde{\mathsf{vk}}_j$, and core C_j has access to $\widetilde{\mathsf{vk}}_i$ and vk_j, with $\widetilde{\mathsf{vk}}_i$, $\widetilde{\mathsf{vk}}_j$ being appropriate sanitations of vk_i, vk_j using the same randomness. For this reason, we additionally define an appropriate generic algorithm that allows to combine these keys either way to output the same value.

Definition 7 (Consistent identity combinability)**.** *A sanitizable signature scheme has consistent identity combinability if it supports an algorithm* IDComb *with the following property:*

$$\mathsf{IDComb}(\mathsf{vk}_i, \mathsf{MaulVK}(\mathsf{vk}_j, \rho)) = \mathsf{IDComb}(\mathsf{MaulVK}(\mathsf{vk}_i, \rho), \mathsf{vk}_j)$$

5.1.5 Construction from BLS.

We report the BLS signature scheme [8] in the following.

- $\mathsf{KeyGen}(1^\lambda) = (\mathsf{sk}, \mathsf{vk}) = (x, g^x)$
- $\mathsf{Sign}(\mathsf{sk}, m) = H(m)^{\mathsf{sk}}$
- $\mathsf{Vrfy}(\mathsf{vk}, (m, \sigma))$: Check $\hat{e}(\sigma, g) = \hat{e}(H(m), \mathsf{vk})$

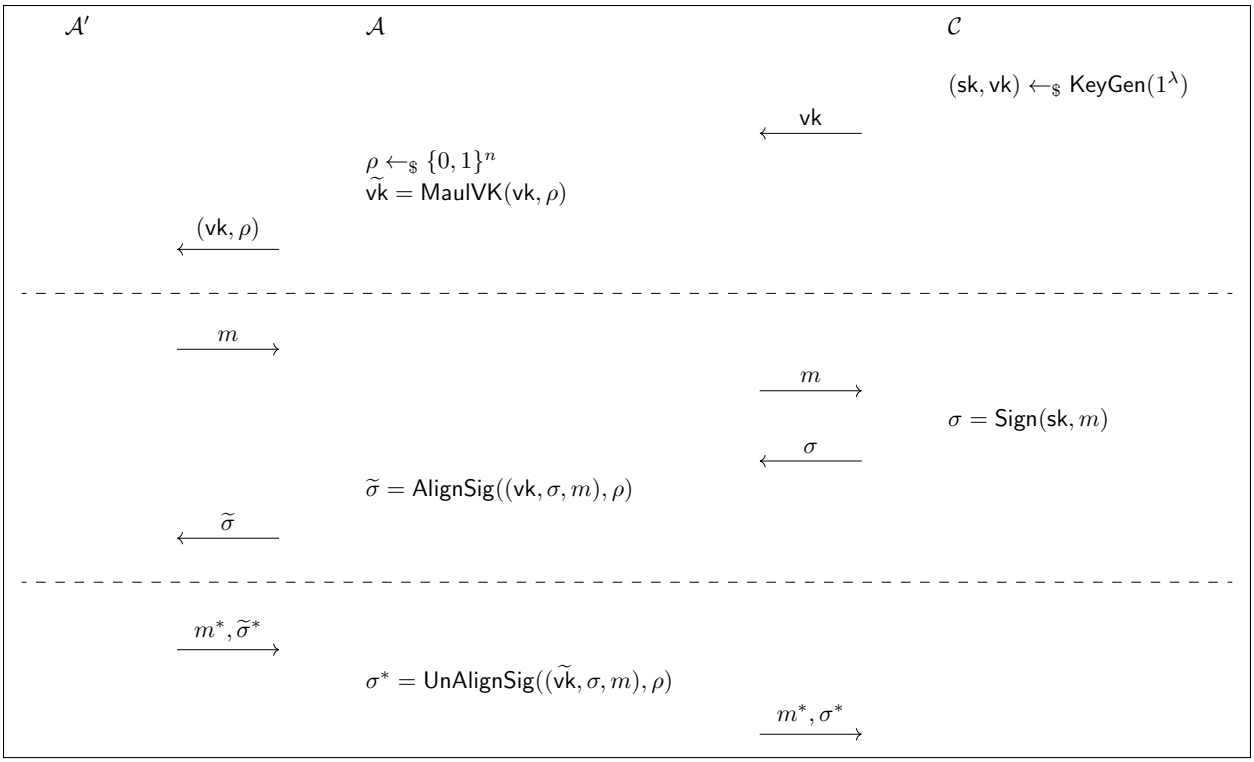

Figure 5: A reduction proving Theorem 5. $\mathcal{A}'$ is the algorithm who breaks SAT EUF-CMA with non-negligible probability. $\mathcal{A}$ acts as the challenger for $\mathcal{A}'$ and as the adversary for EUF-CMA. $\mathcal{C}$ is the challenger for EUF-CMA.

The BLS signature scheme is already compatible with all the additional interfaces required by a sanitizable signature scheme, and has the consistent identity combinability property:

- $\mathsf{MaulVK}(\mathsf{vk}, \rho) = \mathsf{vk}^\rho$
- $\mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho) = \sigma^\rho$
- $\mathsf{UnAlignSig}((\mathsf{vk}, \widetilde{\sigma}, m), \rho) = \widetilde{\sigma}^{\rho^{-1}}$
- $\mathsf{IDComb}(\mathsf{vk}_i, \mathsf{vk}_j) = \hat{e}(\mathsf{vk}_i, \mathsf{vk}_j)$

Theorem 6. *The BLS signature scheme [8] with the additional algorithms specified above is a sanitizable signature scheme with consistent identity combinability, assuming that H is a random oracle and that CDH is hard for $\mathbb{G}$.*

Proof. It has already been shown that the BLS signature scheme is EUF-CMA [8]. Using Theorem 5, we know that it is also SAT EUF-CMA as defined in Definition 6. It remains to show that the signature scheme has consistent key sanitation and alignment invertibility, as formalized in Definition 5.

Consistent key sanitation is immediate from inspection. Denoting $(\mathsf{sk}, \mathsf{vk}) = \mathsf{KeyGen}(1^\lambda) = (x, g^x)$, $\sigma = \mathsf{Sign}(\mathsf{sk}, m) = H(m)^x$ with $m \in \{0,1\}^n$, $\widetilde{\mathsf{vk}} = \mathsf{MaulVK}(\mathsf{vk}, \rho) = g^{x\rho}$, $\widetilde{\sigma} = \mathsf{AlignSig}(\mathsf{vk}, \sigma, m, \rho) = (H(m)^x)^\rho$, the verification of signature $\widetilde{\sigma}$ for message m under $\widetilde{\mathsf{vk}}$ (i.e., $\mathsf{Vrfy}(\widetilde{\mathsf{vk}}, (\widetilde{\sigma}, m)) : \hat{e}(\widetilde{\sigma}, g) \stackrel{?}{=} \hat{e}(H(m), \widetilde{\mathsf{vk}})$) succeeds:

$$\begin{aligned}\hat{e}(\widetilde{\sigma}, g) &= \hat{e}(H(m)^{x\rho}, g) \\ &= \hat{e}(H(m), g^{x\rho}) \\ &= \hat{e}(H(m), \widetilde{\mathsf{vk}})\end{aligned}$$

Alignment invertibility also follows by inspection:

$$\mathsf{UnAlignSig}(\widetilde{\mathsf{vk}}, \widetilde{\sigma}, m, \rho) = (H(m)^{x\rho})^{\rho^{-1}} = H(m)^x = \sigma$$

Finally, we show consistent identity combinability:

$$\begin{aligned}\mathsf{IDComb}(\mathsf{vk}_i, \mathsf{MaulVK}(\mathsf{vk}_j, \rho)) &= \hat{e}(g^{x_i}, g^{x_j\rho}) \\ &= \hat{e}(g^{x_i\rho}, g^{x_j}) \\ &= \mathsf{IDComb}(\mathsf{MaulVK}(\mathsf{vk}_i, \rho), \mathsf{vk}_j)\end{aligned}$$

□

5.1.6 Realizing $s\mathcal{F}_{\mathsf{SAT}}$.

We now describe a protocol that realizes $s\mathcal{F}_{\mathsf{SAT}}$ in the 2-party setting. The link initialization phase is depicted in Figure 6, and the message authentication phase in Figure 7. In particular:

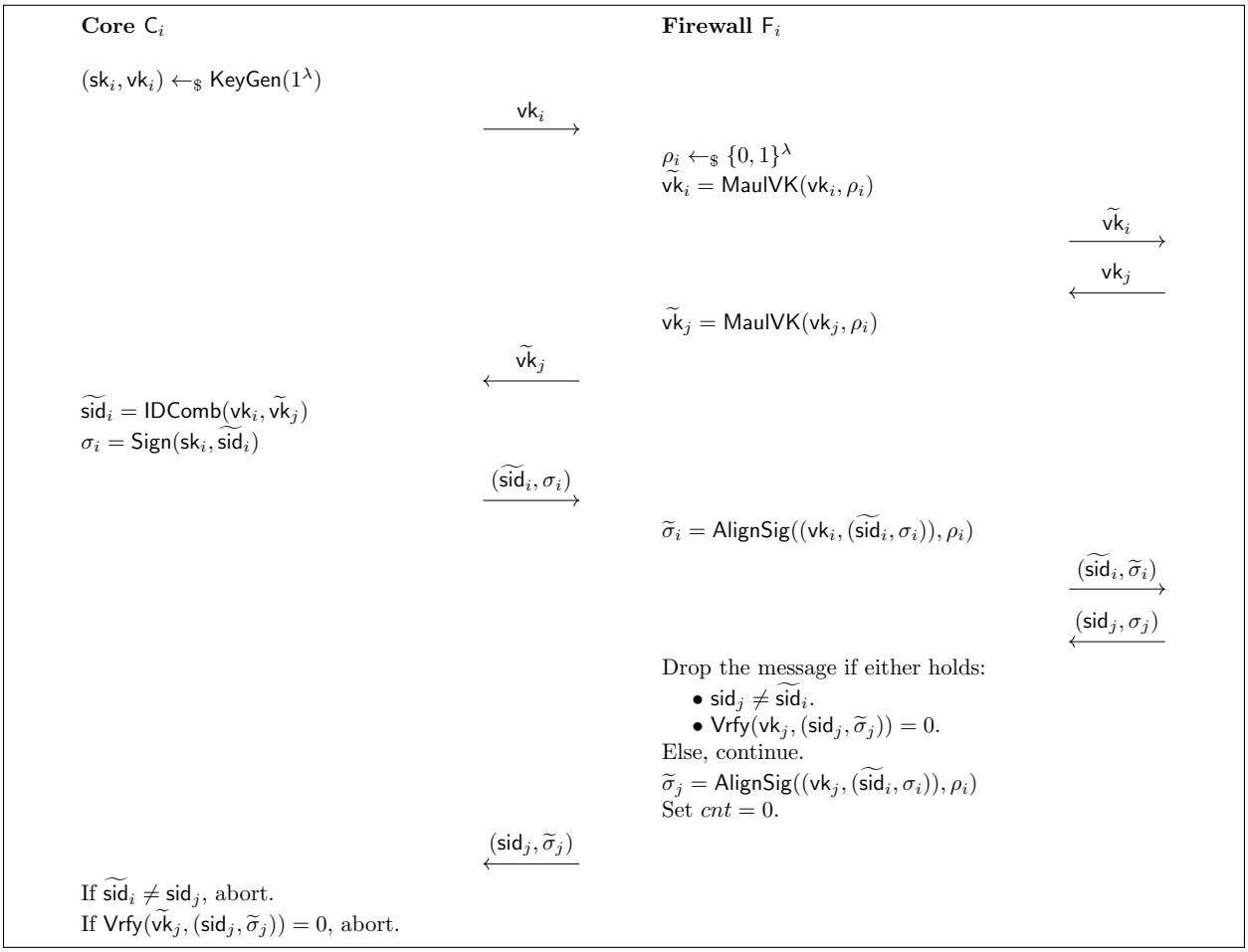

Figure 6: Diagram of the protocol implementing the link initialization phase of $s\mathcal{F}_{\mathsf{SAT}}$.

- In the link initialization phase, the firewall blinds the (identifying) verification key vk_i to $\widetilde{\mathsf{vk}}_i$ using blinding factor ρ_i, and forwards it (supposedly) to party P_j. Upon receiving the verification key

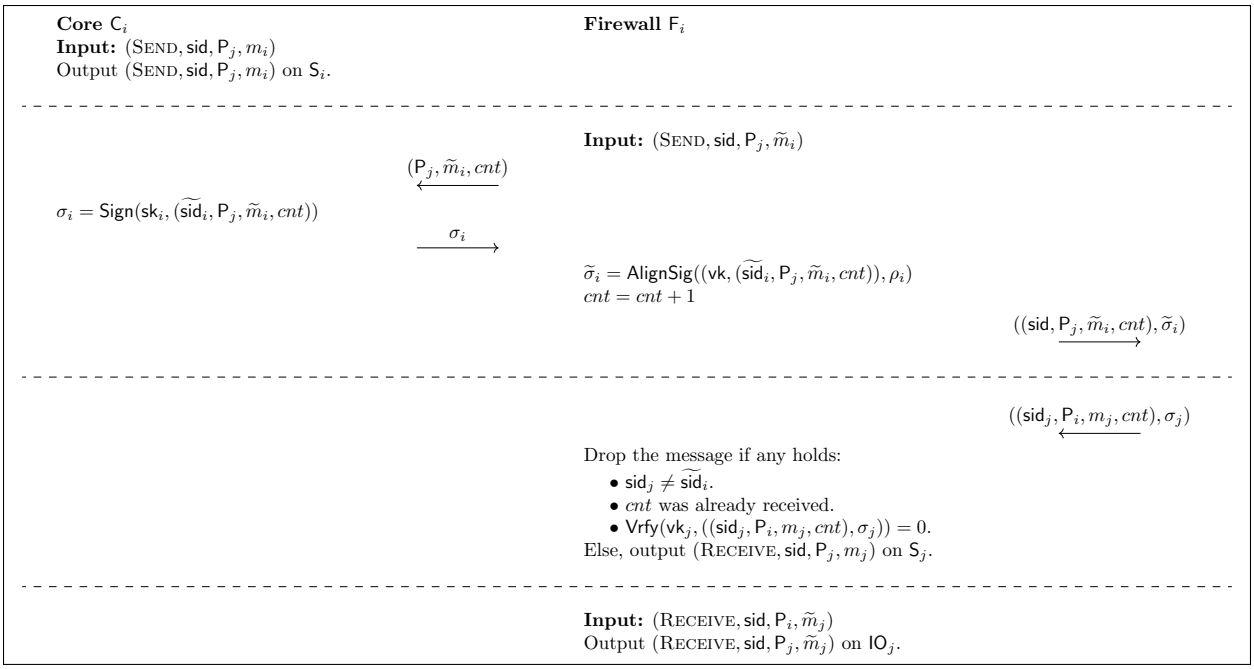

Figure 7: Diagram of the protocol implementing the message authentication $s\mathcal{F}_{\mathsf{SAT}}$, split in each of the interfaces.

vk_j (supposedly) from party P_j, the firewall blinds it to $\widetilde{\mathsf{vk}}_j$ with ρ_i and returns it to its core. The core then uses sk_i to sign its SID $\widetilde{\mathsf{sid}}_i$, computed from vk_i and $\widetilde{\mathsf{vk}}_j$ in such a way that it is equal to an SID computed from $\widetilde{\mathsf{vk}}_i$ and vk_j (using the consistent identity combinability property of the sanitizable signature scheme). Then, the firewall aligns the signature to verify under verification key $\widetilde{\mathsf{vk}}_i$ and outputs it (supposedly) to party P_j, along with $\widetilde{\mathsf{sid}}_i$. Whenever the firewall receives a message $(\mathsf{sid}_j, \sigma_j)$, it first checks whether sid_j is consistent with $\widetilde{\mathsf{sid}}_i$, and whether the signature verifies. If not, the message is dropped, as it may be input triggering. Otherwise, it aligns σ_j to $\widetilde{\sigma}_j$ with ρ_i and forwards $(\mathsf{sid}_j, \widetilde{\sigma}_j)$ to its core, who checks whether $\widetilde{\mathsf{sid}}_i = \mathsf{sid}_j$ (i.e., both parties agree on a consistently computed SID), and whether the signature is valid. If so, the message authentication phase may be carried out.

- In the message authentication phase, we distinguish the four interfaces in different interactions:

  1. Whenever the core of the calling protocol inputs a message (Send, $\mathsf{sid}, \mathsf{P}_j, m$) on IO_i, C_i outputs the message on S_i (*i.e.*, to the firewall of the calling protocol).
  2. Whenever the firewall of the calling protocol inputs a message (Send, $\mathsf{sid}, \mathsf{P}_j, \widetilde{m}$) on S_i, F_i sends the message back to its core C_i, who produces a signature σ_i using secret key sk_i and forwards σ_i to F_i. F_i then aligns the signature to $\widetilde{\sigma}_i$ using the same ρ_i of the link initialization phase, updates the counter and sends the message and the signature (supposedly) to party P_j.
  3. Whenever F_i receives a message $\widetilde{m}$ (supposedly) coming from party P_j, it first checks whether the signature is valid, whether the SID matches the one of the core, and whether a message with the same counter was already sent. If any check fails, the message is dropped. Otherwise, it is output on S_i (*i.e.*, to the firewall of the calling protocol).
  4. Whenever the firewall of the calling protocol inputs a message (Receive, sid, P_j, $\hat{m}$) on S_i, C_i outputs the message on IO_i (*i.e.*, to the core of the calling protocol).

Theorem 7. *The protocol depicted in Figures 6, 7 realizes the $s\mathcal{F}_{\mathsf{SAT}}$ functionality, assuming a sanitizable EUF-CMA signature scheme with consistent identity combinability and the presence of secure channels between cores and their respective firewall.*

We first show that the firewalls are *strongly sanitizing*, meaning that the environment is unable to distinguish a protocol run in which either core is speciously corrupted and their firewall is honest from a run in which the same core is incorruptible and their firewall is honest.

Lemma 7. *The firewall F_i in Figure 3, is strongly sanitizing.*

Proof (Lemma 7). We show that no environment $\mathcal{E}$, which allows specious corruptions for the core C_i but does not corrupt the respective firewall F_i, is able to distinguish between an execution of the protocol in Figures 6, 7 with an incorruptible C_i from one that accepts the corruption $\tilde{\mathsf{C}}_i$. We analyze the link initialization and the message authentication phases separately.

Link Initialization. We first argue that no input-trigger attacks are possible. The inbound values for C_i are vk_j and $(\mathsf{sid}_j, \sigma_j)$, all of which are sanitized as follows:

- vk_j is sanitized to a fresh verification key by means of algorithm MaulVK with randomness ρ_i.
- sid_j is publicly verifiable and it matches a known sanitized value, so any different message is dropped (in particular, it should match the outbound $\widetilde{\mathsf{sid}}_i$).
- σ_j is sanitized to a new signature using the same blinding factor ρ_i.

We now show that C_i is unable to leak any information. The core sends a verification key vk_i and session ID $\widetilde{\mathsf{sid}}_i$ with a matching signature σ_i, all of which are sanitized as follows:

- vk_i is sanitized to a fresh verification key by means of algorithm MaulVK with randomness ρ_i.
- $\widetilde{\mathsf{sid}}_i$ is publicly verifiable and is a sanitized value, so any different message is dropped.
- σ_i is sanitized to a new signature using the same blinding factor ρ_i.

Message Authentication. The only inbound message to C_i is the tuple $(\mathsf{P}_j, \tilde{m}_i, cnt)$, which is output by the honest F_i, so no triggering exists.

The only message outbound from C_i is the signature σ_j, which gets sanitized by means of algorithm $\mathsf{AlignSig}$ with the same randomness ρ_i used in the link initialization phase.

The sanitation of all other messages (both inbound and outbound) is taken care of by the calling protocol. □

Lemma 8. *For every adversary $\mathcal{A}$ corrupting any subset of parties maliciously and the firewall of honest parties semi-honestly in an execution of the protocol Π as in Figures 6, 7, there exists a simulator $\mathcal{S}$ such that, for all environments $\mathcal{E}$, assuming secure channels between cores and their firewall and a sanitizable EUF-CMA signature scheme with consistent identity combinability, the following holds:*

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

Proof (Lemma 8). The proof runs similarly to the one of Barak et al. (Proof of Theorem 11, [5]): the simulator $\mathcal{S}$ partitions parties according to the SID they compute, and the unforgeability of the signature scheme prevents the adversary from forcing inconsistent authentication sets among honest parties or injecting arbitrary messages in the channel.

Intuitively, the main differences consist in (i) the core combining verification keys to a single value, (ii) the adversary controlling the blinding factor used by the (semi-honest) firewall to sanitize the verification key and to align signatures, (iii) the presence of additional communication between a core and its firewall. Item

(i) is covered by consistent identity combinability. Item (ii) is covered by assuming a sanitizable EUF-CMA signature scheme, so knowing the blinding factor does not help towards producing a valid forgery. Item (iii) is covered by the assumption of secure channels between a core and its firewall.

We refer to the 2-party setting in which both parties are composed of an honest core and a semi-honest firewall. The simulator internally simulates the behaviour of the honest cores in the link initialization phase by producing keys vk_i and vk_j respectively. The simulator also stores keys $\widetilde{\mathsf{vk}}_j$, $\widetilde{\mathsf{vk}}_i$ received from $\mathcal{A}$. Now, the simulator checks whether $\mathsf{IDComb}(\mathsf{vk}_i, \widetilde{\mathsf{vk}}_j) = \mathsf{IDComb}(\widetilde{\mathsf{vk}}_i, \mathsf{vk}_j)$ (i.e., whether the locally-received keys have been blinded consistently), and whether the signatures generated on $\widetilde{\mathsf{sid}}_i$ and $\widetilde{\mathsf{sid}}_j$ verify. If so, it places P_i and P_j in the same authentication set H. In any other case, it places P_i and P_j in different authentication sets.

Compared to the original proof, the output of IDComb now immediately identifies the identification set and its related SID in a correct manner (as we are in the 2-party case) due to the consistent identity combinability property.

For what concerns the simulation of communication between firewalls, the strategy is the same as the original proof. The only way their simulation fails in the message authentication phase is by having a RECEIVE message from party P_j to party P_i coming out of the blue from $\mathcal{A}$. This occurs only by considering two honest cores with semi-honest firewalls, as otherwise the signature would have been generated by $\mathcal{S}$. In this case, if $\mathcal{A}$ forges a signature on behalf of the honest core C_j, even by knowing the randomness used to maul the respective verification key, it may be used to break sanitizable EUF-CMA.

The communication between cores and their firewall is immediate, due to the presence of a secure channel between said components: SEND or RECEIVE messages coming from $s\mathcal{F}_{\mathsf{SAT}}$ (on behalf of dummy components) are forwarded to the appropriate component; SEND from (corrupted) cores or RECEIVE from (corrupted) firewalls messages are forwarded to $s\mathcal{F}_{\mathsf{SAT}}$. □

Proof (Theorem 7). The theorem statement follows by looking at the standard corruption translation table. Since both parties are honest, they are either composed of (i) a HONEST core and a SEMIHONEST firewall, or (ii) a SPECIOUS core and a HONEST firewall. Since both firewalls are strongly sanitizing, as shown in Lemma 7, the core in case (ii) can be considered HONEST (Lemma 2 of [15]). Hence, the statement follows directly by Lemma 8. We consider ISOLATE corruptions as MALICIOUS corruptions. □

5.2 Multi-Session Ideal Functionalities in srUC

In this section we first recap the notion of a UC "multi-session" functionality $\mathcal{F}$, and then we show that this notion can be cast in the srUC model. Finally, we prove that in the srUC model with $\mathcal{F}_{\mathsf{SAT}}$ any functionality can be poly-realized.

Informally, a multi-session ideal functionality in UC is an ideal functionality that allows "multiple runs" of the functionality using the *same setup*. As a concrete example, the commitment functionality $\mathcal{F}_{\mathsf{COM}}$ allows a committer to commit to a single value; to produce another commitment a new and independent instance of (the protocol realizing) $\mathcal{F}_{\mathsf{COM}}$ must be spawn with a brand new setup. In contrast, the multi-session functionality $\mathcal{F}_{\mathsf{MCOM}}$ allows a committer to perform poly-many commitments using the same setup. More formally, in UC we say that a protocol π n-emulates a protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any n-instance environment $\mathcal{E}$ we have that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{E}} \approx \text{EXEC}_{\phi,\mathcal{S},\mathcal{E}}$. An n-instance environment is one that interacts with n concurrent instances of the protocol, *i.e.*, it is allowed to invoke ITIs with up to n different SIDs.

Casting this notion into the srUC model, one simply needs to consider the same definition as above but replacing the n-instance environment $\mathcal{E}$ with a *specious* n-instance environment $\mathcal{E}'$ and replacing the adversary $\mathcal{A}$ with a *specious* adversary $\mathcal{A}'$. Now we are ready to state the lemma.

Lemma 9. *For any regular (well-formed)*[5] *ideal functionality $\mathcal{F}$ there exists a protocol π that n-realizes $\mathcal{F}$ in the wsrUC model assuming authenticated channels in the presence of static and malicious adversaries for $n = poly(\lambda)$. Moreover, the protocol π is such that all instances of π use a single instance of $\mathcal{F}_{\mathsf{crs}}$.*

[5]The "well-formed" property is to rule out unrealistic functionalities as explained in [12, 5].

Proof (sketch). We follow the proof of [5] adjusting to our setting when necessary. Let $\Phi_{\mathcal{F}}$ be the protocol of [15] realizing functionality $\mathcal{F}$ in the wsrUC model. The structure of $\Phi_{\mathcal{F}}$ can roughly be broken down into two main components: (1) a protocol $\pi_{\hat{\mathcal{F}}_{\mathsf{C\&P}}}$ for realizing a multi-instance commit-and-prove functionality $\hat{\mathcal{F}}_{\mathsf{C\&P}}$ that provides multiple independent commit-and-prove operations making (implicit) use of a single CRS functionality $\mathcal{F}_{\mathsf{crs}}$, and (2) a protocol $\Pi_{\mathcal{F}}$ for realizing functionality $\mathcal{F}$ assuming access to $\hat{\mathcal{F}}_{\mathsf{C\&P}}$. We can express the protocol $\Phi_{\mathcal{F}}$ as $\Phi_{\mathcal{F}} = \Pi_{\mathcal{F}}^{\pi_{\hat{\mathcal{F}}_{\mathsf{C\&P}}}}$. Observe that $\pi_{\hat{\mathcal{F}}_{\mathsf{C\&P}}}$ produces each commit-and-prove operation in a separate subroutine, and the only state that is shared is the access to the CRS. Thus, we conclude that for any n, running n independent instances of $\Phi_{\mathcal{F}}$ where each instance uses a different instance of $\hat{\mathcal{F}}_{\mathsf{C\&P}}$ is exactly the same as running the same n instances of $\Phi_{\mathcal{F}}$ where all instances use a single joint instance of $\hat{\mathcal{F}}_{\mathsf{C\&P}}$. Then, we can invoke the composition theorem of the srUC model and replace the the joint instance of $\hat{\mathcal{F}}_{\mathsf{C\&P}}$ by a single instance of a protocol that realizes $\hat{\mathcal{F}}_{\mathsf{C\&P}}$. We end up with a protocol that realizes n independent instances of $\mathcal{F}$ and uses a single instance of $\mathcal{F}_{\mathsf{crs}}$. □

5.3 Realizing Generic Split Functionalitites

In this section we finally show that any protocol π that realizes a 2-party functionality $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model (*i.e.*, using authenticated channels) can be compiled into a protocol Π that realizes the split 2-party functionality $s\mathcal{F}$ in the $s\mathcal{F}_{\mathsf{SAT}}$-hybrid model (*i.e.*, using unauthenticated channels).

5.3.1 Description of a Generic Split Functionality.

Below we give a formal description of a generic split functionality. Informally, a split functionality is a wrapper of a functionality $\mathcal{F}$ where the adversary can separate parties (each made of a core and a firewall) into disjoint execution sets.

Functionality $s\mathcal{F}$

$s\mathcal{F}$ is a *split* sanitizable ideal functionality that interacts with a set of parties $\mathcal{P}$, each composed of a core C and a firewall F, and an adversary $\mathcal{S}$. The functionality consists of the following communication interfaces for the cores and the firewalls respectively.

Initialization

- Upon activation with input $(\textsc{Init}, \mathsf{sid})$, parse $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$ where $\mathcal{P}$ is a set of parties, record $(\mathcal{P}, \mathsf{sid}')$ and forward it to the adversary $\mathcal{S}$.
- Upon receiving the message $(\textsc{Init}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$, from $\mathcal{S}$: Verify that party $\mathsf{P} = (\mathsf{C}, \mathsf{F}) \in \mathcal{P}$, that the list H of party identities includes P, and that for all recorded sets H', either $H \cap H'$ contains only corrupted parties (as defined in the standard corruption transition table) and $\mathsf{sid}_H \neq \mathsf{sid}_{H'}$, or $H' = H$ and $\mathsf{sid}_H = \mathsf{sid}_{H'}$. If any check fails, do nothing. Otherwise, output $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ to P and initialize an instance of the ideal functionality $\mathcal{F}$ with SID sid_H, denoted as $\mathcal{F}_H$. The adversary plays the role of parties $\mathcal{P} - H$ in $\mathcal{F}_H$, and any sub-components of the remaining (honest) parties in H can be corrupted consistently with the standard corruption translation table (i.e., specious cores or semi-honest firewalls).

Computation

- Upon receiving an input $(\textsc{input}, \mathsf{sid}, v)$ from a honest component of party $\mathsf{P} \in \mathcal{P}$ on any interface $\mathsf{I} \in \{\mathsf{IO}, \mathsf{S}\}$, find the set H such that $\mathsf{P} \in H$, and forward the message v from P to $\mathcal{F}_H$ on interface I.
- Upon receiving an input $(\textsc{input}, \mathsf{sid}, H, \mathsf{P}, \mathsf{comp}, v)$ from $\mathcal{S}$, for $\mathsf{comp} \in \{\mathsf{C}, \mathsf{F}\}$, if $\mathcal{F}_H$ is initialized and either $\mathsf{P} \in \mathcal{P} - H$ or $\mathsf{P} \in \mathcal{P}$ but comp is a corrupted component, forward v to $\mathcal{F}_H$ as if coming

from comp on interface IO if comp = C (resp. interface S if comp = F). Otherwise, ignore the message.

- When an instance $\mathcal{F}_H$ generates an output v for a honest component of party $\mathsf{P} \in H$ on any interface $\mathrm{I} \in \{\mathsf{IO}, \mathsf{S}\}$, output v on I. When the output is for a party $\mathsf{P} \in \mathcal{P} - H$, for a corrupted component of party $\mathsf{P} \in \mathcal{P}$, or for $\mathcal{S}$, send the output to $\mathcal{S}$.

5.3.2 Compiler for Split Protocols.

Barak et al. [5] show a compiler for transforming a protocol π realizing functionality $\mathcal{F}$ in the UC $\mathcal{F}_{\mathsf{MAUTH}}$-hybrid model into a protocol Π realizing functionality $s\mathcal{F}$ in the UC $\mathcal{F}_{\mathsf{SA}}$-hybrid model. This result can be mapped to our framework by replacing $\mathcal{F}_{\mathsf{MAUTH}}$ with $\mathcal{F}_{\mathsf{SAT}}$, and $\mathcal{F}_{\mathsf{SA}}$ with $s\mathcal{F}_{\mathsf{SAT}}$, with the additional detail that messages coming from $s\mathcal{F}_{\mathsf{SAT}}$ are forwarded to the instance of the protocol π on the respective interface (i.e., IO or S), rather than having a single interface for each party.

Lemma 10. *Let $\mathcal{G}$ be a setup functionality, let $\mathcal{F}$ be an 2-party ideal functionality, and let π be a protocol that securely 2-realizes $\mathcal{F}$ in the wsrUC model with authenticated communication (*i.e.*, $\mathcal{F}_{\mathsf{SAT}}$) and a single instance of $\mathcal{G}$. Then, protocol Π realizes the split functionality $s\mathcal{F}$ in the wsrUC model using a single instance of $s\mathcal{F}_{\mathsf{SAT}}$ and a single instance of $\mathcal{G}$.*

Proof (sketch). The idea of the proof is to show that for any specious adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no specious environment $\mathcal{E}$ can tell with non-negligible probability whether it is interacting with adversary $\mathcal{A}$ and parties running protocol Π or with the simulator $\mathcal{S}$ and the ideal functionality $s\mathcal{F}$. For that, we simply extend the proof of [5, Lemma 4.2] to account for the srUC model. The outline of the proof of [5] in the UC framework is as follows: (1) Given an adversary $\mathcal{A}$ that interacts with (a single instance of) protocol Π on *unauthenticated* channels construct an adversary $\mathcal{A}'$ that interacts with (up to n concurrent instances of) protocol π on *authenticated* channels. (2) Since π securely n-realizes $\mathcal{F}$ then there exists a simulator $\mathcal{S}'$ such that the output of any UC n-instance environment $\mathcal{E}'$ that interacts with $\mathcal{S}'$ and multiple instances of $\mathcal{F}$ is indistinguishable from the output of $\mathcal{E}'$ when interacting with $\mathcal{A}'$ and π. (3) Transform the simulator $\mathcal{S}'$ into a simulator $\mathcal{S}$ for the ideal execution of $s\mathcal{F}$. (4) Finally, given an environment $\mathcal{E}$ (that interacts with $\mathcal{A}$ and Π), construct an n-instance environment $\mathcal{E}'$ (that interacts with $\mathcal{A}'$ and π) such that $\mathcal{E}'$ distinguishes between an interaction with $\mathcal{A}'$ and π from an interaction with $\mathcal{S}'$ and $\mathcal{F}$ with the same probability that $\mathcal{E}$ distinguishes between an interaction with $\mathcal{A}$ and Π from an interaction with $\mathcal{S}$ and $s\mathcal{F}$.

The main difference between the standard UC model and the srUC model is that in srUC the environment and the adversary can be *specious*. Namely, a specious environment is one that upon its first activation writes $(\textsc{Specious}, \mathsf{D})$ on a special tape, where D is an efficiently sampleable distribution of specious cores. 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.

One crucial point in the proof is to note that environment $\mathcal{E}'$ is specious whenever $\mathcal{E}$ is specious; since $\mathcal{E}'$ is constructed black-box from $\mathcal{E}$, it runs $\mathcal{E}$ as its first step, and thus correctly samples and writes $(\widetilde{\mathsf{C}}_1, \ldots, \widetilde{\mathsf{C}}_m, a)$ to a special tape as the first thing it does. The second point is that if $\mathcal{A}$ is specious then $\mathcal{A}'$ is also specious, since $\mathcal{A}'$ will run an instance of $\mathcal{A}$ internally and can relay any specious corruption instruction. As our instantiation of $s\mathcal{F}_{\mathsf{SAT}}$ is only for the 2-party case, the lemma must be cast to the specialized 2-party setting. □

Putting it all together. We show that the split functionalities notion of [5] can be cast in the subversion-resilient UC model in the same way as in standard UC. Namely, one can build protocols for the authenticated channel setting and simply invoke Theorem 4 for claiming security of the split version of the protocol in the unauthenticated channel setting (albeit only for 2-party functionalities).

6 Sanitizing PAKE

So far we have only realized the $\mathcal{F}_{\mathsf{RE}}$ functionality, in which the adversary is unable to perform any (online) password guesses. Similarly to the work of Canetti *et al.* [10], we can immediately obtain $\mathcal{F}_{\mathsf{PAKE}}$ by weakening authenticated channels among firewalls just to be split authenticated, using the results we presented in Section 5.

Corollary 1. *The protocol in Figure 3 wsrUC-realizes the $s\mathcal{F}_{\mathsf{RE}}$ ideal functionality in the $(\mathcal{F}_{\mathsf{sOT}}, s\mathcal{F}_{\mathsf{SAT}})$-hybrid model under static corruptions.*

Corollary 1 follows from Theorem 4, which allows us to replace $\mathcal{F}_{\mathsf{SAT}}$ with $s\mathcal{F}_{\mathsf{SAT}}$ in the protocol so that it instantiate the split version of randomized equality $s\mathcal{F}_{\mathsf{RE}}$. Finally, we remark that $s\mathcal{F}_{\mathsf{RE}}$ is equivalent to $\mathcal{F}_{\mathsf{PAKE}}$, as shown in prior work such as [19, 5].

7 Conclusions

We presented the first subversion-resilient UC protocol for PAKE. We formalized oblivious transfer in the subversion setting, and extended the framework to the unauthenticated setting, providing an implementation for its respective backbone of communication (i.e., $s\mathcal{F}_{\mathsf{SAT}}$) in the two-tier model without assuming a PKI. Finally, we instantiated $\mathcal{F}_{\mathsf{PAKE}}$ by replacing, in a sanitized protocol for $\mathcal{F}_{\mathsf{RE}}$, the $\mathcal{F}_{\mathsf{SAT}}$ assumption with $s\mathcal{F}_{\mathsf{SAT}}$. Several interesting research questions remain open, such as fully instantiating $\mathcal{F}_{\mathsf{SAT}}$ in the two-tier model, expanding the notion of split functionalities in the srUC model to the n-party setting, extending the framework to adaptive corruptions, weakening trusted setups to be subvertable, and achieving explicit mutual authentication for randomized equality and PAKE.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377. Springer, Heidelberg, August 2005.

[6] 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.

[7] 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.

[8] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

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

[10] Ran Canetti, Dana Dachman-Soled, Vinod Vaikuntanathan, and Hoeteck Wee. Efficient password authenticated key exchange via oblivious transfer. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 449–466. Springer, Heidelberg, May 2012.

[11] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005.

[12] 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.

[13] 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.

[14] Suvradip Chakraborty, Chaya Ganesh, Mahak Pancholi, and Pratik Sarkar. Reverse firewalls for adaptively secure MPC without setup. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 335–364. Springer, Heidelberg, December 2021.

[15] Suvradip Chakraborty, Bernardo Magri, Jesper Buus Nielsen, and Daniele Venturi. Universally composable subversion-resilient cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 272–302. Springer, Heidelberg, May / June 2022.

[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.

[17] 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.

[18] 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.

[19] Pierre-Alain Dupont, Julia Hesse, David Pointcheval, Leonid Reyzin, and Sophia Yakoubov. Fuzzy password-authenticated key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 393–424. Springer, Heidelberg, April / May 2018.

[20] 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.

[21] 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.

[22] 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.

[23] Adam Groce and Jonathan Katz. A new framework for password-based authenticated key exchange. Cryptology ePrint Archive, Report 2010/147, 2010. https://eprint.iacr.org/2010/147.

[24] 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.

[25] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

A Basics of the UC framework

We briefly recap the UC framework from [9]. A protocol Π 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 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 P_i for purely idiomatic reasons.

A party 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 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 $\mathsf{Exec}_{\Pi,\mathcal{A},\mathcal{E}}$ 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 [9].

The protocol Π 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 $\mathsf{Exec}_{\mathcal{F},\mathcal{S},\mathcal{E}}$, where we do not specify the dummy protocol but use $\mathcal{F}$ for the dummy protocol composed with $\mathcal{F}$. We say that Π UC-realizes $\mathcal{F}$ if there exists an efficient simulator which makes the simulation look like the real world execution to any efficient environment:

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

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 Π 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 Γ 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 Π, 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 Γ. Similarly, any output produced by the protocol Γ is treated as a value provided by the functionality $\mathcal{G}$. The composition theorem states that in such a case, Π and $\Pi^{\mathcal{G}\to\Gamma}$ have essentially the same input/output behavior. Namely, Γ behaves just like the ideal functionality $\mathcal{G}$ even when composed with an arbitrary protocol Π. A special case of this theorem states that if Π UC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, then $\Pi^{\mathcal{G}\to\Gamma}$ UC-realizes $\mathcal{F}$.

B Non-sanitized protocols and ideal functionalities

In this section, we report some of the descriptions for ideal functionalities and non-sanitized variants of protocols that are already known in the literature.

B.1 Split Authentication

Functionality $\mathcal{F}_{\mathsf{SA}}$

$\mathcal{F}_{\mathsf{SA}}$ is an ideal functionality that interacts with a set of parties $\mathcal{P}$ and an adversary $\mathcal{S}$.

Initialization

Upon receiving query $(\textsc{Init}, \mathsf{sid})$, **where** $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$**:**

Record $\mathcal{P}$ and forward it to $\mathcal{S}$.

Upon receiving query $(\textsc{Init}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$, **from** $\mathcal{S}$**:**

Verify that party $\mathsf{P} \in \mathcal{P}$, that the list H of party identities includes P, and that for all recorded sets H', either $H \cap H'$ contains only corrupted parties and $\mathsf{sid}_H \neq \mathsf{sid}_{H'}$, or $H' = H$ and $\mathsf{sid}_H = sid_{H'}$. If so, output $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ to P and record (H, sid_H) if not yet recorded. Else, do nothing.

Message authentication

Upon receiving query $(\textsc{Send}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$, **where** $\mathsf{P} \in \mathcal{P}$**:**

Send $(\mathsf{P}, \mathsf{P}', m)$ to $\mathcal{S}$ and add $(\mathsf{P}, \mathsf{P}', m)$ to an (initially empty) list $\mathcal{W}$ of waiting messages. The same triple can appear multiple times in the list.

Upon receiving query $(\textsc{Deliver}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$ **from** $\mathcal{S}$**:**

- If P' did not previously receive an $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ output then do nothing.
- Else, if P is in the authentication set H of P', and P is uncorrupted, then: if there is a triple $(\mathsf{P}, \mathsf{P}', m) \in \mathcal{W}$, remove one appearance of the triple from $\mathcal{W}$ and output $(\textsc{Received}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$ to P'. Otherwise do nothing.
- Else (i.e., P' received $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$, and either P is corrupted or $\mathsf{P} \notin H$), output $(\textsc{Received}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$ to P', regardless of $\mathcal{W}$.

As shown in Barak *et al.* [5], the protocol depicted in Figure 8 instantiates $\mathcal{F}_{\mathsf{SA}}$.

B.2 Oblivious Transfer

We report the generic framework of Peikert *et al.* [25] in Figure 9.

B.3 Randomized Equality

We depict in Figure 10 a variant of the construction of RE from OT of Canetti *et al.* [10] using a 1-bit password.

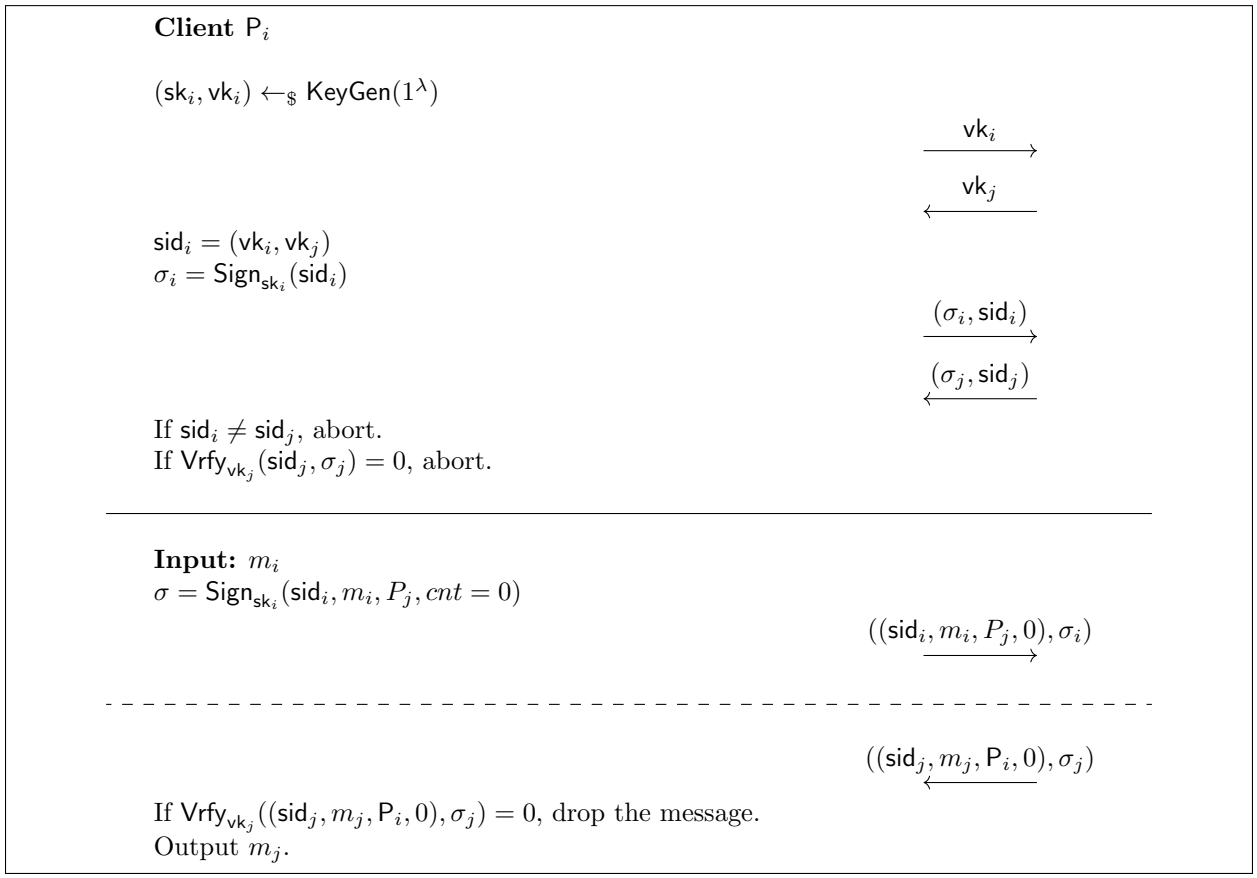

Figure 8: Session-authenticated channel of Barak et al. [5]. On the top half, the setup phase. On the bottom half, the message authentication phase, consisting of sending a message, and receiving and verifying an inbound message.

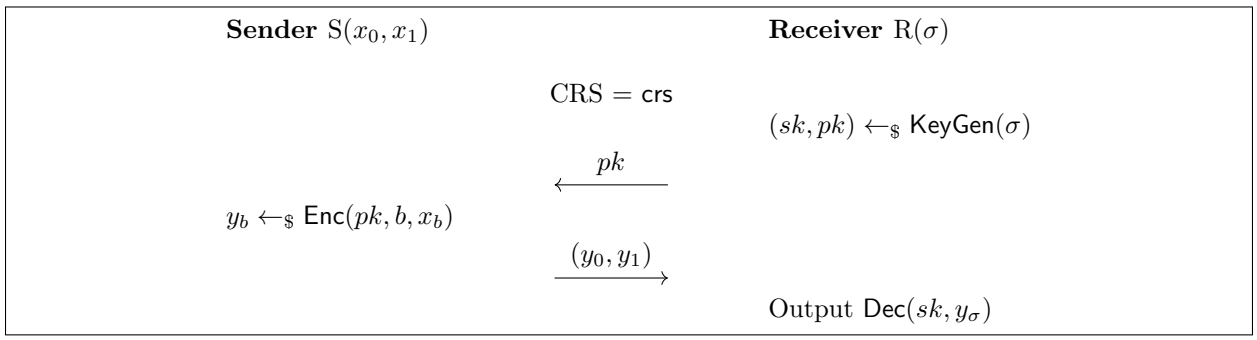

Figure 9: The generic framework for $\mathcal{F}_{\mathsf{OT}}$ of Peikert et al. [25].

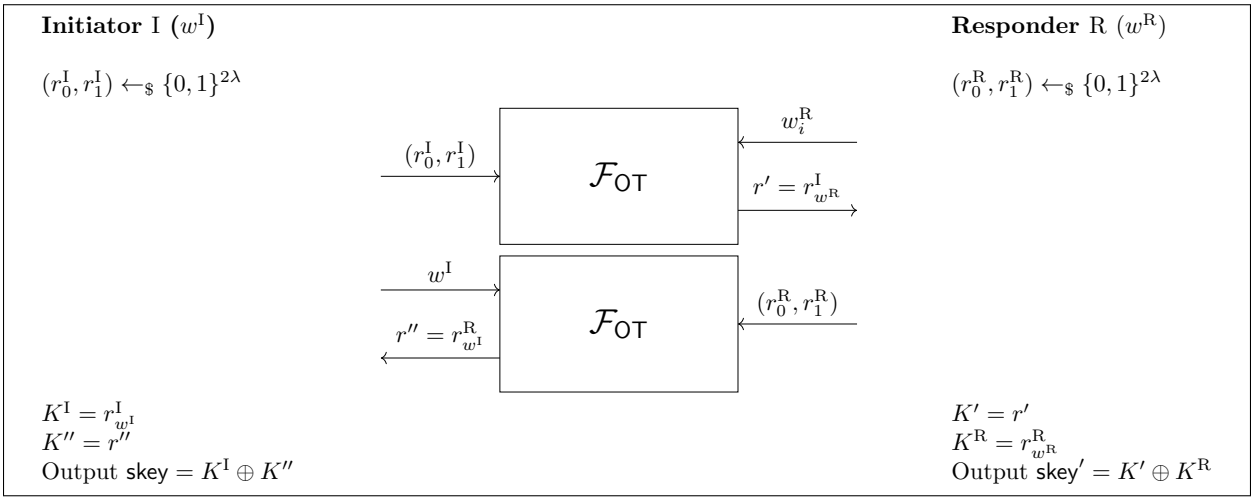

Figure 10: $\mathcal{F}_{\mathsf{RE}}$ from OT [10] with a 1-bit password.