Key Exchange in the Post-Snowden Era: Universally Composable Subversion-Resilient PAKE*

Suvradip Chakraborty¹, Lorenzo Magliocco^{†2}, Bernardo Magri³, and Daniele Venturi^{†2}

¹VISA Research suvradip1111@gmail.com ²Sapienza University of Rome magliocco@di.uniroma1.it, venturi@di.uniroma1.it ³University of Manchester and Primev bernardo.magri@manchester.ac.uk

November 24, 2024

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., EU-ROCRYPT'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

^{*}Full version of an extended abstract that will appear in Proceedings of Asiacrypt 2024, Springer-Verlag, 2024. Available from the IACR Cryptology ePrint Archive as Report 2023/1827.

[†]Lorenzo Magliocco and Daniele Venturi were supported by project SERICS (PE00000014) and by project PARTHENON (B53D23013000006), under the MUR National Recovery and Resilience Plan funded by the European Union—NextGenerationEU. Daniele Venturi is member of the Gruppo Nazionale Calcolo Scientifico - Istituto Nazionale di Alta Matematica (GNCS-INdAM).

1 Introduction

Authenticated Key Exchange (AKE) allows two parties to generate a shared high-entropy secret and mutually authenticate 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 an honest party could have been compromised in an undetectable manner, both in the case of its hardware (e.g., via backdoored components) or its software (e.g., via algorithm-substitution attacks, purposefully designed leaky constructions, or mistakenly instantiated protocols). Such undetectable corruptions enable an adversary to launch so-called subversion attacks, which may cause the target compromised machine to covertly exfiltrate information or behave in an unexpected manner upon receiving a specific triggering input.

A possible mitigation consists in equipping parties with *cryptographic reverse firewalls* (RFs), as first defined by Mironov and Stephens-Davidowitz [27]. 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 of [27] 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 [18]. The latter ensures that, once a designed protocol is proven to be secure, subversion resilience holds even if that protocol is arbitrarily composed with other protocols. This lifts the requirement of redoing the security analysis from scratch for each individual composition setting, 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 (srUC) framework [18], in which parties can derive a high-entropy secret key and verify their identities by means of a shared 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 outright. 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 subversion attacks, via RFs. Following [18], 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 shares no secrets with the core and sanitizes the outgoing/incoming communication from/to the core using random coins). Both the core and the RF are subject to different flavours of corruption, modelling different kinds of subversion attacks.

In order to avoid simple impossibility results, we follow [18] and only consider the so-called *specious* subversions, in which a subverted core looks like an honest core to any efficient test, yet it may signal private

information to the subverter via subliminal channels, or trigger an unexpected behaviour whenever a specific triggering message is received.

Our PAKE protocol is obtained by sanitizing the UC randomized equality protocol from oblivious transfer (OT) by Canetti et al. [12]. As an added bonus, this construction allows us to introduce several building blocks of independent interest in the srUC framework in a modular and natural manner. As we explain in the next section, essential changes to the original building blocks' design are needed, including the definition and the realization of sanitizable variants of intermediate ideal functionalities, new sanitation-friendly properties for cryptographic primitives, and extensions to the srUC model itself.

One difficulty in the realization of PAKE is that one cannot rely on authenticated channels. As shown by Barak et al. [7], 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 randomized equality, which is essentially PAKE with authenticated channels, assuming the existence of a functionality for sanitizable authenticated communication (which already appeared in [18], and is denoted by \mathcal{F}_{SAT}). Following [7], we then define a weaker split-authenticated (sanitizable) variant $s\mathcal{F}_{SAT}$ that allows the adversary to partition parties, and prove that a modification of their transformation allows to lift any protocol that multi-realizes a functionality \mathcal{F} assuming authenticated channels to one that realizes the corresponding "split version" (i.e., $s\mathcal{F}$) without any assumption on channels, even in the presence of subversion.

In the process, we realize $s\mathcal{F}_{\mathsf{SAT}}$ by sanitizing the protocol of [7, Section 4.2], introducing a new notion of key-sanitizable signature schemes with a matching security property. This improves on an open problem from [18], 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., $s\mathcal{F}_{\mathsf{SAT}}$).

Finally, we apply the aforementioned transformation to our randomized equality protocol, and realize subversion-resilient PAKE by constructing a protocol with access to the split version of the randomized equality functionality.

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}_{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.

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

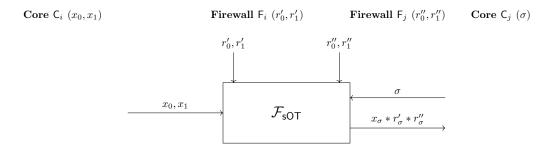


Figure 1: Our sanitizable OT functionality \mathcal{F}_{sOT} , with * being an appropriate blinding operation for the input domain.

overall party related to that firewall is malicious. 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: once we have mapped components to a malicious party we shouldn't simulate anything that occurs within that malicious party. As an example, while handling a malicious sender in a protocol realizing \mathcal{F}_{sOT} , it suffices for the simulator to only forward to \mathcal{F}_{sOT} the malicious sender's input messages. Indeed, the notion of blinding may not even be well-defined.

In order to instantiate \mathcal{F}_{sOT} , we start by considering dual-mode cryptosystems as in Peikert et al. [28]. Briefly, in these cryptosystems the party holding the secret key specifies a decryption branch upon generating the keypair, and the party holding the public key specifies an encryption branch for each ciphertext. Decryption succeeds only for ciphertexts generated on the decryption branch. Moving to the subversion setting, we introduce a new primitive that we call sanitizable homomorphic dual-mode cryptosystems that extends dual-mode cryptosystems by additionally providing: (1) a procedure to carry out homomorphic operations on ciphertexts (e.g., $\text{Enc}(m_1) * \text{Enc}(m_2) = \text{Enc}(m_1 * m_2)$), (2) a procedure to maul an encryption key pk to a different encryption key pk, and (3) a procedure to maul a ciphertext under encryption key pk to a ciphertext of the same message under encryption key pk. Looking ahead, item (1) allows firewalls to sanitize the messages input to the OT, and items (2, 3) allow to first blind a 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 [28, Section 5] 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 [28, Section 4], 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 and sends the public key towards the sender. This key is sanitized once by each firewall. Upon receiving the (sanitized) key, the sender encrypts value x_b on encryption branch b, for $b \in \{0, 1\}$, and forwards these ciphertexts towards the receiver. Each firewall removes one layer of sanitation from the ciphertexts, so that the receiver can successfully decrypt the ciphertext 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, as it leverages on the two (computationally indistinguishable) modes of the CRS to map the behaviour of the adversary to consistent queries to \mathcal{F}_{sot} .

We conclude the section by remarking that, exactly as in the original protocol of [28], it is possible to re-use the same CRS across multiple protocol runs. Hence, we obtain a protocol that multi-realizes \mathcal{F}_{sOT} (i.e., a protocol that realizes the multi-session sanitizable OT functionality $\hat{\mathcal{F}}_{sOT}$).

1.3.2 Sanitizing Randomized Equality

Canetti et al. [12] instantiate the randomized equality functionality by proposing a protocol that relies on OT and roughly unfolds as follows: for an n-bit password, each party runs \mathcal{F}_{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 in the i-th run. Intuitively, the sender of each batch of OTs is able to choose the same random strings that were selected by the receiver only if the passwords are the same, and all these strings can be combined to derive a common shared key.

After defining \mathcal{F}_{sOT} , designing a protocol that realizes the randomized equality ideal functionality \mathcal{F}_{RE} in the subversion setting becomes immediate. In order to thwart information leakage originating from a biased sampling of the random 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.

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.

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.~[7]. 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, while in this model it is not possible to achieve the same guarantees as with authenticated channels, meaningful security can still be provided: namely, the worst the adversary can do is split honest parties into independent execution sets before the protocol run, and act on behalf of all (honest) parties that are not within the same set. This way, even though honest parties can run the entire protocol with the adversary without even noticing it, they can rest assured that they will complete the entire run of the protocol interacting with the same set of parties since the start. In [7], this notion is captured in what the authors call split functionalities. One central result of [7] consists of showing a generic transformation for which any protocol UC n-realizing some n-party functionality $\mathcal F$ relying on authenticated channels can be compiled into a similar protocol that UC-realizes the split functionality $s\mathcal F$, but now just relying on u-mauthenticated channels.

Given that [18] exclusively refers to authenticated channels, which are formalized with the "sanitizable authenticated transmission" functionality $\mathcal{F}_{\mathsf{SAT}}$, in this work we extend the notion of split functionalities to the srUC model. More specifically, we show that the generic transformation of [7] for split protocols carries over to our setting whenever the underlying unauthenticated channel is sanitizable. The latter notion is captured by the split version of $\mathcal{F}_{\mathsf{SAT}}$, that we call $s\mathcal{F}_{\mathsf{SAT}}$. This functionality allows the adversary to split parties in different authentication sets in a "link initialization" phase, before any message is exchanged. After that, the behaviour is exactly the same as $\mathcal{F}_{\mathsf{SAT}}$, except that the adversary may deliver arbitrary messages to parties within different authentication sets.

A crucial component of the transformation is the construction of a protocol realizing $s\mathcal{F}_{\mathsf{SAT}}$. For that, we introduce a new primitive that we call key-sanitizable signatures that: (1) provides a procedure to maul a verification key vk into vk , (2) a procedure to maul a signature under verification key vk into a signature under verification key vk , and (3) is equipped with a function f such that $f(\mathsf{vk}_i, \mathsf{vk}_j) = f(\mathsf{vk}_i, \mathsf{vk}_j)$, with vk_i and vk_j being verification keys mauled under the same randomness. We show that the BLS signature scheme [10] is a key-sanitizable signature scheme, with f being a bilinear map. In our protocol for $s\mathcal{F}_{\mathsf{SAT}}$, parties exchange locally-generated keys, which are used to "initialize the link" by determining a session ID sid, and to sign messages that are exchanged through the link. Firewalls sanitize these keys and re-align signatures accordingly to preserve correctness, and the bilinear map allows parties to recompute the same sid in the presence of firewalls mauling the keys. We note however that the bilinear map restricts the protocol to the 2-party setting, which in turn 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, one can simply white-box inspect the proofs of [7] and adapt them to the srUC setting. The core result is a lemma stating that any protocol 2-realizing a 2-party functionality \mathcal{F} in the wsrUC model assuming $\mathcal{F}_{\mathsf{SAT}}$ can be compiled into a protocol realizing $s\mathcal{F}$ in the wsrUC model assuming $s\mathcal{F}_{\mathsf{SAT}}$. Given that any n-party functionality \mathcal{F} can be n-realized in the wsrUC model by the subversion-resilient GMW compiler of [18], we also obtain a theorem stating that any 2-party split functionality can be realized in the wsrUC model using only unauthenticated channels (in the $s\mathcal{F}_{\mathsf{SAT}}$ -hybrid model), matching [7, Theorem 10]. As in traditional UC, a protocol poly-realizing a functionality roughly means that polynomially-many instances of that protocol may re-use the same setup.

1.3.4 The final PAKE protocol

At last, we combine all our ingredients together to realize PAKE in the subversion setting. First, we apply the split transformation to the protocol realizing \mathcal{F}_{RE} in the authenticated setting, obtaining a protocol that realizes $s\mathcal{F}_{RE}$ in the unauthenticated setting. Then, with a similar argument to that of Dupont *et al.* [22], we argue that $s\mathcal{F}_{RE}$ is sufficient to instantiate \mathcal{F}_{PAKE} . This can be shown by exhibiting a trivial protocol in the $s\mathcal{F}_{RE}$ -hybrid model that exclusively interacts with $s\mathcal{F}_{RE}$, and by showing that the power of splitting parties in $s\mathcal{F}_{RE}$ can be mapped to the power of performing password queries in \mathcal{F}_{PAKE} .

We observe that, as a corollary of the generic result of the previous paragraph, one also gets a protocol realizing $s\mathcal{F}_{RE}$ by relying on the srUC GMW compiler from [18], although with worse efficiency than our concrete construction from DDH. For that, we provide a hand-wavy comparison of the two constructions by considering communication and round complexity.

Importantly, in this work we consider a PAKE functionality that only provides implicit rather than explicit authentication. This means that, while parties can be assured by the functionality that any other party capable of deriving the same session key must possess the password, there is no direct assurance that the counterpart has successfully computed the session key upon completion of the protocol. This decision was made for two primary reasons: (1) it streamlines our results, as explicit mutual authentication typically requires incorporating additional "key confirmation" steps at the protocol's conclusion, which would complicate our protocol with the need for further sanitation processes, and (2) in many practical scenarios, such as secure channels, explicit authentication is not a requirement. Moreover, in our setting, mutual explicit authentication is inherently provided by any higher-level protocol that utilizes our PAKE as a foundation. For instance, in applications involving secure messaging, the act of successfully exchanging messages serves as explicit confirmation that both parties share the same session key.

Moreover, as a technical remark stemming from the srUC model, the PAKE functionality we realize implicitly includes the wrapper of [18] that simply adds dummy firewall parties in order to prevent trivial distinguishing from the environment. This also holds for \mathcal{F}_{RE} , but causes no differences in the behaviour of both functionalities. For a cleaner presentation and following [18], we omit the wrapper when using hybrid functionalities.

1.4 Related Work

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

1.4.1 Reverse firewalls and subversion

Reverse firewalls were introduced by Mironov and Stephens-Davidowitz [27], 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 [21, 19], interactive proof systems [24], and maliciously secure MPC for both the case of static [16] and adaptive [17] 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 [18] 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. [18] shows how to sanitize the classical GMW compiler [25] for MPC under subversion. Towards that, it also introduces the concept of sanitizable commitment and sanitizable commit-and-prove.

More recently and concurrently to this work, an alternative framework for subversion-resilient UC was put forward by Arnold et al. [4]. Compared to [18], this new framework captures reverse firewalls in the plain UC model, but characterizes subversion by exclusively allowing an adversary to tamper with the function generating the randomness of a protocol. This rules out simple subversion attacks which [18] (and our paper) accounts for, such as having a specious core change its input to part of its secret state upon receiving a specific triggering value.

Ringerud [29] explored the problem of achieving subversion-resilient AKE in a standalone fashion (*i.e.*, without reverse firewalls or watchdogs), providing intuition on why realizing this primitive appears to be hard in such an adversarial setting.

Additional work on subversion includes algorithm substitution attacks [9, 6, 20], parameter subversion [8, 2, 23, 3], Cliptography [5, 32, 31], subliminal channels [33, 34] to list a few. We refer to [27, 18] for further related works, such as watchdogs and self-guarding.

1.4.2 PAKE

The seminal work by Canetti et al. [13] 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 [11], i.e., when protocols can be arbitrarily composed with other protocols. The description was later extended to explicit mutual authentication in [26] and [12], in which parties are able to tell whether they effectively authenticated or not. Our work is the first to achieve subversion-resilient PAKE in the UC framework.

1.5 Organization

In Section 2, we give a concise introduction to the subversion-resilient UC framework of [18]. In Section 3, we define and instantiate sanitizable oblivious transfer. In Section 4, we instantiate a subversion-resilient 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. [7] that allows to remove authenticated channels from our reference framework. In Section 6, we combine the results of previous sections to achieve subversion-resilient PAKE. Finally, in Section 7, we conclude the paper with a few related open problems for further research. See Figure 2 for a visual representation of how our results are linked to one another.

2 A Brief Recap of Subversion-resilient UC

We give a brief overview of subversion resilience in the UC framework (srUC for short). We refer the reader to [18] for further details, and to [11] for a complete treatment of the UC framework. We give a succinct introduction of the latter in Appendix B.

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 [18] specifies that the relevant corruption cases for the core are HONEST, MALICIOUS, or Specious, while the ones for the firewall are

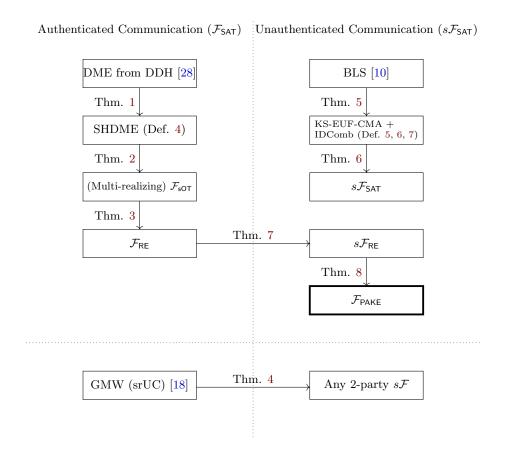


Figure 2: A visual summary of the contributions of this paper. All the functionalities are realized in the srUC framework of [18]. DME stands for Dual-Mode Encryption. SHDME stands for Sanitizable Homomorphic DME. KS-EUF-CMA stands for Key-Sanitizable EUF-CMA. IDComb is a shorthand for Consistent Identity Combinability.

HONEST, SEMIHONEST, or MALICIOUS. Mapping the corruption possibilities for the parties $P_i = (C_i, F_i)$ in a regular UC functionality gives rise to the following corruption translation table:

2.1.1 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{C}_i which cannot be distinguished from C_i by black-box access to \widetilde{C}_i or C_i . Intuitively, a specious corruption can be thought of as a subversion that remains undetectable.

2.1.2 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 in the authenticated setting, and as a MITM attack in the unauthenticated setting, and can therefore be safely dropped from the analysis.

Firewall F	Party P in \mathcal{F}
SemiHonest	Honest
Honest	Honest
Malicious	ISOLATE
Malicious	Malicious
	SemiHonest Honest Malicious

Table 1: The corruption translation table of [18].

2.1.3 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 [18], whenever the firewalls are strongly sanitizing, the SPECIOUS core and HONEST firewall case is the same as considering an HONEST core and an HONEST firewall.

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. For what concerns security definitions, two separate notions are presented in [18], according to the type of functionality that is being realized: subversion-resilient UC (srUC) security for sanitizable ideal functionalities, and wrapped subversion-resilient UC (wsrUC) security for regular ideal functionalities. Definition 1 captures security of a sanitizing protocol 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 $\mathsf{Wrap}(\mathcal{F})$ in the \mathcal{G} -hybrid model. This yields wsrUC security. Definition 2 is essentially the same, but captures security for protocols realizing a sanitizable ideal functionality, yielding srUC security.

Definition 1 (Wrapped subversion-resilient UC security [18]). Let \mathcal{F} be an ideal functionality for n parties P_1, \ldots, P_n . Let Π be a sanitizing protocol with n cores C_1, \ldots, C_n and n firewalls F_1, \ldots, 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 $\operatorname{Wrap}(\mathcal{F})$ in the \mathcal{G} -hybrid model, with the restriction that we only quantify over specious environments and specious adversaries.

Definition 2 (Subversion-resilient UC security [18]). Let \mathcal{F} be an ideal functionality for n parties P_1, \ldots, P_n . Let Π be a sanitizing protocol with n cores $C_1^{\mathcal{F}}, \ldots, C_n^{\mathcal{F}}$ and n firewalls $F_1^{\mathcal{F}}, \ldots, F_n^{\mathcal{F}}$. 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

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

functionality², 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 the protocols of [18], communication is mediated by a sanitizable ideal functionality for authenticated communication \mathcal{F}_{SAT} , which fundamentally embeds three capabilities:

- 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.

In what follows, we report a variant of the description of \mathcal{F}_{SAT} that does *not* include the first capability. This is a design choice that allows to better separate setup and communication: indeed, the former may be captured by a separate ideal functionality \mathcal{F}_{crs} .

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

- On input (Send, P_i , P_j , a) 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 (Send, P_i , P_k , b) on S_i , it leaks the tuple to the adversary S, and internally stores the tuple.
- On input (Deliver, (Send, P_i , P_k , b)) from the adversary, where the Send tuple is stored, it outputs (Receive, P_i , P_k , b) on S_k and deletes the tuple.
- On input (RECEIVE, P_i , P_m , c) on S_m , it outputs (RECEIVE, P_i , P_m , c) on IO_m .

An important observation is that $\mathcal{F}_{\mathsf{SAT}}$ induces a core-to-core authenticated channel. While this is an acceptable backbone of communication for our protocols in Sections 3 and 4, it makes little sense to instantiate PAKE by already assuming authenticated channels. In Section 5, we overcome this limitation by defining a weaker functionality $s\mathcal{F}_{\mathsf{SAT}}$ that models the unauthenticated setting by allowing the adversary to partition parties, in line with the work of Barak *et al.* [7].

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, define sanitizable homomorphic dual-mode cryptosystems, and exhibit an instantiation for this new primitive from the DDH assumption. We use the latter notion to sanitize the generic framework for OT of Peikert et al. [28], obtaining a protocol for the sanitizable oblivious transfer functionality \mathcal{F}_{sOT} . Finally, we argue that, in line with the instantiation of [28], our protocol can reuse the same CRS across multiple runs, thus realizing the multi-session extension of \mathcal{F}_{sOT} (also denoted by $\hat{\mathcal{F}}_{sOT}$).

3.1 Sanitizable OT

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

²A 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.

Functionality \mathcal{F}_{sOT}

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

Interface IO_i :

Upon receiving a query (SENDER, sid, (x_0, x_1)) from C_S on IO_S :

Record (SENDER, sid , (x_0, x_1)) and forward the tuple on S_i . Ignore subsequent commands of the form (SENDER, sid , ·).

Upon receiving a query (RECEIVER, sid, σ) from C_R on IO_R :

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

- * The message (BLIND, sid, \cdot) was sent to \mathcal{F}_{sot} on S_s . If S is malicious according to the corruption translation table, mark this check as passed.
- * The message (BLIND, sid , ·) was sent to \mathcal{F}_{sOT} on S_R . If R is malicious according to the corruption translation table, mark this check as passed.

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

Interface S_i :

Upon receiving a query (BLIND, sid, (x'_0, x'_1)) from F_S on S_S :

If S is malicious according to the corruption translation table, do nothing. Otherwise, check if a record (SENDER, sid , (x_0, x_1)) exists. If so, update the tuple to (SENDER, sid , $(\tilde{x}_0, \tilde{x}_1)$), with $\tilde{x}_b = x_b * x_b'$. Otherwise, do nothing. Ignore future commands of the form (BLIND, sid , ·) on $\operatorname{S}_{\mathbb{S}}$.

Upon receiving a query $(BLIND, sid, (x''_0, x''_1))$ from F_R on S_R :

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

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

If the conditions above hold, update the tuple to (SENDER, 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 (BLIND, sid , ·) on S_R .

The ideal functionality is parameterized by a blinding operation *, which may be tailored to the input domain 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 from firewalls of parties that, according to the corruption translation table, are malicious. As discussed throughout the technical overview in Section 1.3, this is reasonable: the corruption status of individual components can be determined in advance (as we are in the static setting), and their combined behaviour can be considered as a single party by following the corruption translation table. If the joint party is malicious, we do not have to simulate anything related to messages internally exchanged by the adversary. In particular, the blinding operation may not be well-defined at all.

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*, for which the key generation algorithm takes as an additional input a branch $\sigma \in \{0,1\}$. The party holding the public key can choose either branch $b \in \{0,1\}$ over which to encrypt a message. The party holding the secret key is able to decrypt the ciphertext successfully only if $\sigma = b$. Second, 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 report the formal definition of Dual-Mode Cryptosystems in what follows, and refer the reader to [28] for further details.

Definition 3 (Dual-Mode Cryptosystems [28]). A dual-mode cryptosystem consists of a tuple of algorithms (Setup, KeyGen, Enc, Dec, FindMessy, TrapKeyGen) with the following properties:

- 1. Completeness for decryptable branch: For every $\mu \in \{\text{mes, dec}\}$, every $(\text{crs}, t) \leftarrow_{\$} \text{Setup}(1^{\lambda}, \mu)$, every $\sigma \in \{0, 1\}$, every $(pk, sk) \leftarrow_{\$} \text{KeyGen}(\sigma)$, and every $m \in \{0, 1\}^k$, decryption is correct on branch σ , i.e., $\text{Dec}(sk, \text{Enc}(pk, \sigma, m)) = m$. 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 $\mathsf{SetupMessy}$ and $\mathsf{SetupDec}$ are computationally indistinguishable, i.e., $\mathsf{SetupMessy}_1(1^\lambda) \stackrel{c}{\approx} \mathsf{SetupDec}_1(1^\lambda)$.
- 3. (Messy mode) Trapdoor identification of a messy branch: For every $(\operatorname{crs}, t) \leftarrow_{\$} \operatorname{SetupMessy}(1^{\lambda})$ and every (possibly malformed) pk, FindMessy(t, pk) outputs a branch value $b \in \{0, 1\}$ such that $\operatorname{Enc}(pk, b, \cdot)$ is messy. Namely, for every $m_0, m_1 \in \{0, 1\}^k$, $\operatorname{Enc}(pk, b, m_0) \stackrel{s}{\approx} \operatorname{Enc}(pk, b, m_1)$.
- 4. (Decryption mode) Trapdoor generation of keys decryptable on both branches: For every (crs, t) $\leftarrow_{\$}$ SetupDec(1^{λ}), TrapKeyGen(t) outputs (pk, sk_0, sk_1) such that for every $\sigma \in \{0, 1\}$, $(pk, sk_{\sigma}) \stackrel{\approx}{\approx} \text{KeyGen}(\sigma)$.

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. For that, we formally define *sanitizable homomorphic* dual-mode cryptosystems in what follows.

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

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

Intuitively, MaulPK and AlignEnc are defined as a (symmetric) tuple of algorithms as firewalls will first sanitize the outbound encryption key by running MaulPK with some randomness. Then, upon receiving any ciphertext encrypted under the new mauled public key, the firewall will "strip" the layer of sanitation by using the same randomness used for MaulPK, outputting a ciphertext containing the same message for the non-mauled public key pk.

Remark 1. The MaulPK, AlignEnc, HomOp algorithms are outputting keys and ciphertexts implicitly combining the randomness of their inputs. This is essential 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 with "bad randomness" output by their core.

3.2.2 Instantiation from DDH

We briefly recap the instantiation of dual-mode cryptosystems from DDH of [28, Section 5]. 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.
- KeyGen $(\sigma) = ((g_{\sigma}^r, h_{\sigma}^r), r) = ((pk_1, pk_2), sk) = (pk, sk).$
- $\operatorname{Enc}(pk, m, b) = (g_b^s h_b^t, pk_1^s pk_2^t m) = (c_1, c_2).$
- $Dec(sk, c) = c_2/c_1^r$.

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

- MaulPK (pk, ρ) : Output pk^{ρ} .
- AlignEnc (c, ρ) : Parse $c = (c_1, c_2)$. Output $\tilde{c} = (c_1^{\rho}, c_2)$.
- HomOp (c_0, c_1) : Output c_0c_1 .

Theorem 1. The DDH cryptosystem of [28] with the additional algorithms specified above is a sanitizable homomorphic dual-mode cryptosystem, assuming that DDH is hard for \mathbb{G} .

The theorem follows by inspection of the newly-introduced algorithms. A formal proof is given in Appendix C.1.

3.3 A Generic Framework for Sanitizable OT

As shown in the generic framework of [28, Section 4], 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 $b \in \{0, 1\}$. The receiver will only be able to decrypt the ciphertext on branch $\sigma = b$.

From a high-level perspective, our sanitized protocol leverages homomorphic ciphertexts to blind the sender's inputs and uses 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 3, we depict a protocol run showing only the firewall of the sender, since the firewall of the receiver behaves exactly in the same way.

Theorem 2. The protocol in Figure 3, parameterized by mode \in {mes, dec}, realizes the sanitizable functionality \mathcal{F}_{sOT} in the $(\mathcal{F}_{SAT}, \mathcal{F}_{crs})$ -hybrid model under static corruptions. For mode = mes, the sender's security is statistical and the receiver's security is computational; for mode = dec, the security properties are reversed.

Intuitively, we first show that the firewalls are able to thwart all subversion attacks (both inbound and outbound). Then, we simulate similarly to the original proof, with the twist that we do not have to simulate inputs of malicious parties (as per the considerations in the technical overview). We defer the formal proof to Appendix C.2.

3.4 Multi-session \mathcal{F}_{sOT}

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}_{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}_{COM} must be spawned with a brand new setup. In contrast, the multi-session functionality \mathcal{F}_{MCOM} allows a committer to perform poly-many commitments using the same setup. Hence, using multiple instances of \mathcal{F}_{MCOM} has the same effect as using a single instance of \mathcal{F}_{MCOM} .

$$\operatorname{Core} \, \mathsf{C}_{\mathrm{S}} \, (x_0, x_1) \qquad \operatorname{Firewall} \, \mathsf{F}_{\mathrm{S}} \, (x_0', x_1') \qquad \operatorname{Core} \, \mathsf{C}_{\mathrm{R}} \, (\sigma)$$

$$\operatorname{CRS} = \operatorname{crs} \qquad (sk, pk) \leftarrow_{\$} \operatorname{KeyGen}(\operatorname{crs}, \sigma)$$

$$\overset{\rho^{\mathrm{S}}}{\rho^{\mathrm{S}}} \leftarrow_{\$} \{0, 1\}^{\lambda} \qquad \overset{\rho^{\mathrm{K}}}{\rho^{\mathrm{K}}} = \operatorname{MaulPK}(\operatorname{pk}, \rho^{\mathrm{S}})$$

$$\overset{\tilde{y}_b}{\tilde{y}_b} \leftarrow_{\$} \operatorname{Enc}(\widetilde{pk}, b, x_b') \qquad \overset{\tilde{y}_b'}{\tilde{y}_b'} \leftarrow_{\$} \operatorname{Enc}(\widetilde{pk}, b, x_b') \qquad \overset{\tilde{y}_b'}{\tilde{y}_b'} = \operatorname{HomOp}(\tilde{y}_b, \tilde{y}_b') \qquad \overset{(y_0, y_1)}{\tilde{y}_b} \rightarrow \overset{(y_0, y_1)}{\tilde{y}_b'} \rightarrow \overset{(y_0, y_$$

Figure 3: A sanitation of the generic framework of Peikert et al. [28], realizing \mathcal{F}_{sot} . The receiver's firewall is omitted, as it runs the same code as F_S .

Moving to our case, we note that the generic framework of [28] actually realizes the multi-session version of \mathcal{F}_{OT} (also denoted as \mathcal{F}_{OT}). Given that our protocol in Figure 3 has the same structure as the protocol of [28], we observe that we can reuse the same CRS across multiple runs, each with a distinct sub-session ID. The presence of subverted cores does not impact this property, as the sanitation operated from the firewalls uses independently-sampled random strings for each sub-protocol run.

4 Sanitizing Randomized Equality

In this section, we present our sanitized protocol for the (regular) randomized equality ideal functionality \mathcal{F}_{RE} that relies on authenticated channels (i.e., \mathcal{F}_{SAT}) and \mathcal{F}_{SOT} , following the construction of Canetti et al. [12].

Description of $\mathcal{F}_{\mathsf{RE}}$ 4.1

We describe a variation of the randomized equality ideal functionality \mathcal{F}_{RE} of [12], with technical improvements from Dupont et al. [22].

Functionality \mathcal{F}_{RE}

The functionality \mathcal{F}_{RE} is parameterized by a security parameter λ . It interacts with an initiator $I = (C_I, F_I)$, a responder $R = (C_R, F_R)$, and the adversary S via the following messages:

Upon receiving a query (NEWSESSION, sid, I, R, w^{I}), from I:

Record (I, R, w^I) and send a message (sid, I, R) to S. Ignore all future messages from I.

Upon receiving a query (OK, sid) from S:

Send a message (WAKEUP, sid, I, R) to R. Ignore all future (OK) messages.

Upon receiving a query $(RESPOND, sid, I, R, w^R)$ from R:

- $\begin{array}{l} \bullet \ \ {\rm If} \ w^{\rm R} = w^{\rm I}, \ {\rm choose} \ {\sf skey} \leftarrow_{\$} \{0,1\}^{\lambda} \ {\rm and} \ {\rm store} \ {\sf skey}_{\rm I} = {\sf skey}_{\rm R} = {\sf skey}. \\ \bullet \ \ {\rm If} \ w^{\rm R} \neq w^{\rm I}, \ {\rm then} \ {\rm set} \ {\sf skey}_{\rm I} \leftarrow_{\$} \{0,1\}^{\lambda}, \ {\sf skey}_{\rm R} \leftarrow_{\$} \{0,1\}^{\lambda}. \end{array}$

In both cases, ignore subsequent inputs from R.

Upon receiving a query (NewKey, sid, P, K), P \in {I, R} from \mathcal{S} :

- If any of the following conditions hold, output (sid, K) to party P:
 - P is corrupted.
 - $-w^{I}=w^{R}$, and the peer of P is corrupted.
- Otherwise, output (sid, skey_P) to party P.

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

 \mathcal{F}_{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 a NewSession query with password $w^{\rm I}$. The adversary $\mathcal S$ is notified of this interaction, and controls the delivery timing of the initialization message destined to the receiver. Whenever this occurs, the responder inputs its own password $w^{\rm R}$ with a Respond query, and the functionality prepares the output for both parties depending on whether $w^{\rm R} = w^{\rm I}$ 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 the functionality separately for each party, by means of a NewKey query, in which it also specifies a key K. If either party P is corrupted, or the adversary had input party P's password on behalf of its corrupted peer (*i.e.*, it has successfully guessed the password of honest P), P receives the adversarially-generated key K. In any other case, party P receives whatever has been stored in skey_P .

4.2 Randomized Equality from OT

We sanitize the RE from OT protocol of [12, Section 2.2] by using \mathcal{F}_{sOT} , restricting to implicit mutual authentication as per the considerations in the technical overview. 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}_{sOT} . For ease of exposition, we depict the protocol in Figure 4 assuming 1-bit passwords. The *n*-bit password case runs exactly in the same way except that (i) it uses *n* OTs within the multi-session sanitizable OT functionality $\hat{\mathcal{F}}_{sOT}$, and (ii) it computes keys using operator * with *n* random strings rather than only one.

In order to preserve correctness, we leverage 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 4 wsrUC-realizes the \mathcal{F}_{RE} ideal functionality in the $(\mathcal{F}_{sOT}, \mathcal{F}_{SAT})$ -hybrid model under static corruptions.

Within the proof, we first show strong sanitation of firewalls, and then proceed similarly to [12]. We defer the formal proof and an explicit analysis of correctness to Appendix C.3.

5 Subversion-Resilient Split Functionalities

In this section, we extend the notion of split functionalities of Barak et al. [7] to the srUC framework. Informally, we want to show that, for any well-formed³ regular 2-party⁴ ideal functionality \mathcal{F} , there exists a

 $^{^3}$ The "well-formed" property is to rule out unrealistic functionalities as explained in [15, 7].

⁴We restrict our attention to 2-party functionalities (in contrast to [7]) as the theorem relies on the sanitizable $s\mathcal{F}_{\mathsf{SAT}}$ functionality that we only show how to realize for the 2-party setting.

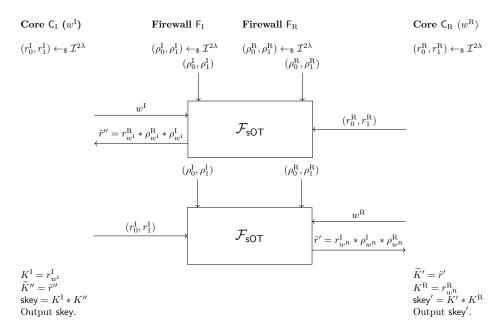


Figure 4: A sanitizing protocol for \mathcal{F}_{RE} from sanitizable OT with a 1-bit password.

protocol that realizes the 2-party $s\mathcal{F}$ functionality with wsrUC-security in the CRS model. More formally, the goal of this section consists in proving an adaptation of [7, Theorem 10] to our setting, *i.e.*:

Theorem 4. Let \mathcal{F} be a (regular) 2-party UC functionality. Then, assuming key-sanitizable signatures with consistent identity combinability, there exists a protocol that securely realizes the 2-party split functionality $s\mathcal{F}$ in the wsrUC model.

Towards that, we follow the same strategy as [7] 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 [18] poly-realizes any functionality in the wsrUC model.
- Unauthenticated channels: Finally, we adapt the generic transformation of [7] that transforms any protocol π that 2-realizes a 2-party functionality \mathcal{F} given authenticated channels (i.e., $\mathcal{F}_{\mathsf{SAT}}$) in the wsrUC model into a protocol that realizes $s\mathcal{F}$ given access to $s\mathcal{F}_{\mathsf{SAT}}$ in the wsrUC 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 [18]) and build a protocol that realizes it in the 2-party setting in the srUC model. For that, we introduce the notion of key-sanitizable signatures and show that it can be instantiated with the BLS signature scheme [10].

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

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. In contrast with $\mathcal{F}_{\mathsf{SAT}}$, the only guarantee provided by the functionality is that each party will be interacting with the same entity throughout the entire protocol run, but that entity could either be the expected party or the adversary itself. 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 an adversary \mathcal{S} and a set of parties, each composed of a core C and a firewall F . The functionality consists of the following communication interfaces.

Initialization

- Upon activation with input (INIT, sid) from party P: Parse $sid = (\mathcal{P}, sid')$ where \mathcal{P} is a set of parties that includes P. Forward (INIT, sid, P) to the adversary \mathcal{S} .
- Upon receiving the message (INIT, $\operatorname{sid}, \mathsf{P}, H, \operatorname{sid}_H$), from \mathcal{S} : Verify that $H \subseteq \mathcal{P}$, that the list H of party identities includes $\mathsf{P} = (\mathsf{C}, \mathsf{F})$, and that for all recorded sets H' either (i) $H \cap H'$ contains only corrupted parties (as per the standard corruption transition table in Table 1) and $\operatorname{sid}_H \neq \operatorname{sid}_{H'}$, or (ii) H' = H and $\operatorname{sid}_H = \operatorname{sid}_{H'}$. If any of the check fails, do nothing. Otherwise, output (INIT, $\operatorname{sid}, \operatorname{sid}_H$) to P and record $(H, \operatorname{sid}_H)$ if not yet recorded.

Message Authentication

- Upon receiving the message (Send, $\mathsf{Sid}, \mathsf{P}_i, \mathsf{P}_j, m$) on IO_i where $\mathsf{P}_j \in \mathcal{P}$: Output the tuple on S_i .
- Upon receiving the message (Send, sid , P_i , \widetilde{m}) on S_i : Add the tuple to an (initially empty) list \mathcal{W} of waiting messages. The same tuple can appear multiple times in the list. Then, leak the tuple to \mathcal{S} .
- Upon receiving the message (Deliver, (Send, sid, P_i , P_j , \tilde{m})) from S:
 - If P_j did not previously receive an (INIT, sid, sid_H) output, 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 (Send, $\operatorname{sid}, P_i, P_j, \widetilde{m}$) $\in \mathcal{W}$, remove one appearance of the tuple from \mathcal{W} and output (Receive, $\operatorname{sid}, P_i, P_j, \widetilde{m}$) on S_j . Otherwise, do nothing.
 - Else (i.e., P_j received (INIT, sid , sid_H), and either P_i is corrupted or $P_i \notin H$), output (RECEIVE, sid , P_i , P_j , \widetilde{m}) on S_j , regardless of W.
- Upon receiving the message (RECEIVE, sid, P_i , P_j , \hat{m}) on S_j , output the tuple on IO_j .

The functionality consists of 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 message is added to a message queue, and 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 is the case for \mathcal{F}_{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 from the adversary.

5.1.2 Key-sanitizable signature schemes

In the construction of \mathcal{F}_{SA} of [7, Section 4.2], 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 *key-sanitizable* signature schemes.

Informally, a key-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 AlignSig to align an (accepting) signature σ produced under secret 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 traditional signatures in Definition 5, introducing a matching security notion in Definition 6 that extends EUF-CMA security to account for the newly introduced algorithms. This new security notion is implied in a black-box manner by any EUF-CMA scheme supporting the aforementioned algorithms.

Definition 5 (Key-sanitizable signature scheme). A key-sanitizable signature scheme consists of a tuple of polynomial-time algorithms (KeyGen, Sign, Vrfy, MaulVK, AlignSig, UnAlignSig) with the following properties:

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

Definition 6 (Key-sanitizable EUF-CMA security). A key-sanitizable signature scheme is key-sanitizable existentially unforgeable against chosen message attacks (KS-EUF-CMA) if the probability of the adversary A winning the following game is negligible:

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

Lemma 1. Any EUF-CMA signature scheme that supports algorithms MaulVK, AlignSig, and UnAlignSig, as defined in Definition 5, is also KS-EUF-CMA.

The proof consists of a black-box reduction to EUF-CMA, and is deferred to the Appendix C.4.

5.1.3 Combining verification keys

Looking ahead, the link initialization phase of the protocol realizing $s\mathcal{F}_{SAT}$ relies on the determination of session IDs via (identifying) verification keys of parties, which get sanitized by firewalls in different directions. For instance, in the 2-party setting, core C_i has access to vk_i and vk_j , and core C_j has access to vk_i and vk_j , with vk_i , vk_j being appropriate sanitations of vk_i , vk_j using the same randomness ρ_i . 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 key-sanitizable signature scheme has consistent identity combinability if it supports an algorithm IDComb with the following property:

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

5.1.4 Instantiation from BLS

We report the BLS signature scheme [10] in the following.

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

The BLS signature scheme is already compatible with all the additional interfaces required by a key-sanitizable signature scheme. Moreover, bilinear maps immediately induce the consistent identity combinability property:

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

Theorem 5. The BLS signature scheme [10] with the additional algorithms specified above is a key-sanitizable signature scheme with KS-EUF-CMA security and consistent identity combinability, assuming that H is a random oracle and that CDH is hard for \mathbb{G} .

The theorem follows by inspecting the newly-introduced algorithms, and by observing that the BLS signature scheme is EUF-CMA. We defer the formal proof to Appendix C.5.

5.1.5 Realizing $s\mathcal{F}_{SAT}$

We now describe a protocol that realizes $s\mathcal{F}_{\mathsf{SAT}}$ in the 2-party setting, which follows a similar structure to that of [7, Section 4.2]. The link initialization phase is depicted in Figure 5, and the message authentication phase in Figure 6.

In the link initialization phase, the firewall blinds the outbound verification key vk_i (identifying party P_i) to vk_i using blinding factor ρ_i , and forwards it (supposedly) to party P_j . Upon receiving the verification key vk_j (supposedly) from party P_j , the firewall blinds it to vk_j with ρ_i and returns it to its core. The core then uses sk_i to sign its SID sid_i , computed from vk_i and vk_j in such a way that it is equal to an SID computed from vk_i and vk_j (using the consistent identity combinability property of the key-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 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, sid, P_i , 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, sid, P_i , 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_i .
- 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_i , \hat{m}) on S_i , C_i outputs the message on IO_i (*i.e.*, to the core of the calling protocol).

Theorem 6. The protocol depicted in Figures 5, 6 realizes the $s\mathcal{F}_{\mathsf{SAT}}$ functionality, assuming a KS-EUF-CMA signature scheme with consistent identity combinability and the presence of secure channels between cores and their respective firewall.

Intuitively, the proof runs as the one for the non-sanitized protocol of [7], except that the blinding operations of firewalls thwart subversion attacks, and consistency between keys is obtained by using IDComb. We defer the formal proof to Appendix C.6.

5.2 Multi-realizing any ideal functionality in the wsrUC model

Next, we prove the following lemma.

Lemma 2. For any regular (well-formed) 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}_{crs} .

Informally, such a protocol can be obtained from the adaptation of the GMW compiler to the srUC framework shown in [18]. The formal proof of the lemma is essentially [7, Theorem 13] verbatim, except that we replace results for the UC framework with their counterparts in the srUC framework, shown in [18] (e.g., the UC composition theorem and the GMW compiler). We defer the formal proof to Appendix C.7.

5.3 Realizing Generic Split Functionalitites

We finally show that any protocol π that wsrUC-2-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 wsrUC-realizes the split 2-party functionality $s\mathcal{F}$ in the $s\mathcal{F}_{\mathsf{SAT}}$ -hybrid model (*i.e.*, using unauthenticated channels). The $s\mathcal{F}$ functionality is exactly the same as in [7]. Indeed, since we wsrUC-realize a regular ideal functionality \mathcal{F} assuming $\mathcal{F}_{\mathsf{SAT}}$, our end goal is to wsrUC-realize the split counterpart of \mathcal{F} assuming $s\mathcal{F}_{\mathsf{SAT}}$, which is also a regular ideal functionality. For completeness, we report a description of $s\mathcal{F}$ in Appendix D.1.

$$\begin{array}{c} \operatorname{Core} \, \mathsf{C}_i \\ (\operatorname{sk}_i, \operatorname{vk}_i) \leftarrow_{\S} \operatorname{\mathsf{KeyGen}}(1^{\lambda}) \\ & \xrightarrow{} \quad \bigvee_{i} \\ & \xrightarrow{$$

Figure 5: Diagram of the protocol implementing the link initialization phase of $s\mathcal{F}_{\mathsf{SAT}}$.

Lemma 3. Let \mathcal{G} be a setup functionality, let \mathcal{F} be a 2-party ideal functionality, and let $\pi_{\mathcal{F}}$ be a protocol that securely 2-realizes \mathcal{F} in the wsrUC model with authenticated communication (i.e., \mathcal{F}_{SAT}) and a single instance of \mathcal{G} . Then, there exists a protocol $\Pi_{\mathcal{F}}$ wsrUC-realizing the split functionality $s\mathcal{F}$ using a single instance of $s\mathcal{F}_{SAT}$ and a single instance of \mathcal{G} .

To prove this theorem, we adapt the proof of [7, Lemma 4.1] to the wsrUC model. First, we describe the protocol $\Pi_{\mathcal{F}}$, which is obtained by adapting the compiler presented in [7]. In particular, the compiler of [7] transforms a protocol $\pi_{\mathcal{F}}$ realizing functionality \mathcal{F} in the UC $\mathcal{F}_{\mathsf{MAUTH}}$ -hybrid model into a protocol $\Pi_{\mathcal{F}}$ realizing functionality $s\mathcal{F}$ in the UC $\mathcal{F}_{\mathsf{SA}}$ -hybrid model. This result can be mapped to our setting by replacing $\mathcal{F}_{\mathsf{MAUTH}}$ with $\mathcal{F}_{\mathsf{SAT}}$, and $\mathcal{F}_{\mathsf{SA}}$ with $s\mathcal{F}_{\mathsf{SAT}}$, with the crucial detail that messages coming from $s\mathcal{F}_{\mathsf{SAT}}$ are forwarded to the instance of the protocol $\pi_{\mathcal{F}}$ on the respective interface (i.e., IO or S), rather than having a single interface for each party. Then, we simply follow the proof of [7, Lemma 4.1] accounting for the additional communication between cores and firewalls and for the presence of specious cores, as per the srUC framework. We defer the description of $\Pi_{\mathcal{F}}$ and the formal proof to Appendix C.8.

5.4 Putting it all together

We showed that the split functionalities notion of [7] can be cast in the subversion-resilient UC model in the same way as in standard UC. Namely, one can build a protocol n-realizing a functionality for the authenticated channel setting and simply invoke Lemma 3 to obtain security of the split version of the protocol in the unauthenticated channel setting (albeit only for 2-party functionalities). Since there exists

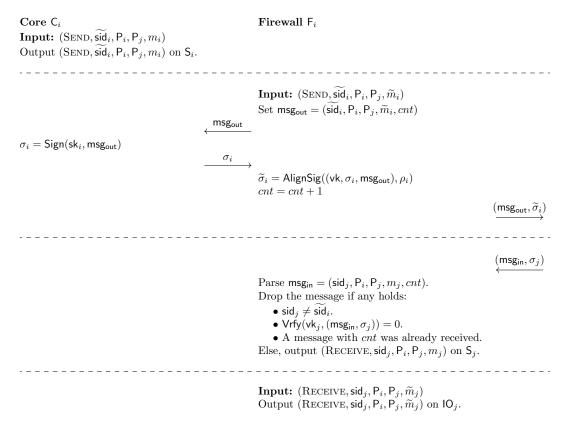


Figure 6: Diagram of the protocol implementing the message authentication $s\mathcal{F}_{SAT}$, split in each of the interfaces.

a protocol 2-realizing *any* regular ideal functionality in the authenticated setting (by using the srUC GMW compiler of [18], as per Lemma 2), there also exists a matching 2-party protocol in the unauthenticated setting realizing the split version of the same functionality, yielding Theorem 4.

6 Sanitizing PAKE

So far we have only referred to the \mathcal{F}_{RE} functionality, in which the adversary is unable to perform any (online) password guesses. In order to move to PAKE, we first provide a description of \mathcal{F}_{PAKE} , highlighting its differences with respect to \mathcal{F}_{RE} . Then, similarly to [12], we argue that our protocol in Section 4 can be compiled in a protocol for $s\mathcal{F}_{RE}$ by invoking a result of Section 5. Finally, we show that $s\mathcal{F}_{RE}$ is sufficient to trivially realize \mathcal{F}_{PAKE} . We conclude the section by highlighting that it is also possible to obtain a protocol for $s\mathcal{F}_{RE}$ by using the general-purpose result given by Theorem 4 (which internally relies on the srUC GMW compiler). In that regard, we provide a hand-wavy performance comparison of such a protocol with our instantiation from DDH.

6.1 Description of $\mathcal{F}_{\mathsf{PAKE}}$

The behaviour of $\mathcal{F}_{\mathsf{PAKE}}$ is conceptually close to that of the $\mathcal{F}_{\mathsf{RE}}$ we described in Section 4.1, with the important difference that the adversary is now allowed to perform (online) password guesses in order to influence the keys output by the functionality. In what follows, we provide a formal description of the $\mathcal{F}_{\mathsf{PAKE}}$ functionality [12] that embeds minor variations to achieve consistency with $\mathcal{F}_{\mathsf{RE}}$, and technical improvements from Dupont *et al.* [22].

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

The functionality \mathcal{F}_{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 (NewSession, sid, I, R, w^{I}) from I:

Record (I, R, w^I) , mark it as fresh, and leak (sid, I, R) to S. Ignore all future messages from I.

Upon receiving a query (OK, sid) from S:

Send a message (WAKEUP, sid, I, R) to R. Ignore all future (OK) messages.

Upon receiving a query (RESPOND, sid, I, R, w^{R}) from R:

Record (R, I, w^R) and mark it as fresh.

Upon receiving a query (TestPwd, sid, P, w') from the adversary S:

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

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

Upon receiving a query (NewKey, sid, P_i , K) from S, where $|K| = \lambda$:

If $P_i \in \{I, R\}$ and there is a record of the form (P_i, P_j, w_i) that is not marked as completed, with P_j being the peer of P_i , then:

- If any of the following conditions hold, output (sid, K) to party P_i :
 - P_i is corrupted.
 - This record is fresh, there exists a record (P_i, P_i, w_i) with $w_i = w_i$, and P_i is corrupted.
 - This record is compromised.
- If this record is fresh, both parties are honest, and there exists a record (P_j, P_i, w_j) with $w_j = w_i$, choose skey $\leftarrow_{\$} \{0, 1\}^{\lambda}$. Output skey to P_i , and append skey to the record (P_i, P_j, w_i) .
- If this record is fresh, both parties are honest, and there exists a record $(P_j, P_i, w_j, \text{skey})$ with $w_j = w_i$, output skey to P_i .
- If none of the above rules apply, choose skey' $\leftarrow_{\$} \{0,1\}^{\lambda}$ and output it to party P_i .

In any case, mark the record (P_i, \cdot, w_i) as completed.

Compared to \mathcal{F}_{RE} , each record now has a status associated to it, which is initially set to fresh. If the adversary performs a password guess against party P's fresh record and fails, the target record is marked as interrupted. If instead the password guess succeeds, the target record is marked as compromised. Either way, the adversary is notified of the outcome of its guess.

Finally, the adversary controls the delivery of the output of the functionality separately for each party, by means of a NewKey query, in which it also specifies a key K. The same (random) key is output to both honest parties only if (i) the input passwords were the same, and (ii) both records were fresh when their respective key was determined (*i.e.*, no password guesses occurred). Otherwise, the functionality outputs the adversarially-generated key if any of the following occurs: (i) P is corrupted, (ii) the adversary does not perform any TestPwD queries but runs the functionality as the peer of P with input the password of P, or (iii) the adversary guesses the password for P via a TestPwD query. In any other case, the functionality outputs a random key to party P.

6.1.1 Variations in the srUC setting

As for \mathcal{F}_{RE} , we restrict our attention to implicit mutual authentication (as discussed in Section 1.3), and the functionality provides no security whatsoever whenever the adversary is able to guess an honest party's password.

6.1.2 Shortcomings of PAKE functionalities

Recent works have raised technical concerns regarding the definition of PAKE functionalities widely used across the literature. Specifically, Abdalla *et al.* [1] observed that several definitions, including the one of the seminal paper of Canetti *et al.* [13], allow the adversary to set the key output by an honest party even without knowing the password. Similarly to Dupont *et al.* [22], our definitions of \mathcal{F}_{RE} and \mathcal{F}_{PAKE} do not embed this shortcoming.

Additionally, Roy and Xu [30] show an impossibility result proving that any 2-party $\mathcal{F}_{\mathsf{PAKE}}$ may be instantiated by an incorrect 0-round protocol. In order to overcome this limitation, they show that either (i) the underlying PAKE protocol is assumed to be correct; (ii) the simulator gets limited in power; or (iii) a third party responsible for routing messages is introduced in $\mathcal{F}_{\mathsf{PAKE}}$. For this work, we solve this shortcoming by considering approach (i), following the spirit of discarding "trivial protocols" in the context of UC (e.g., the empty protocol), as discussed by Canetti et al. [14].

6.2 From $\mathcal{F}_{\mathsf{RE}}$ to $\mathcal{F}_{\mathsf{PAKE}}$

The protocol we presented in Section 4 realizes \mathcal{F}_{RE} in the presence of subversion attacks in the authenticated setting. Proceeding as [12], we convert it to a protocol for $s\mathcal{F}_{RE}$, obtaining the following theorem:

Theorem 7. There exists a protocol that wsrUC-realizes the $s\mathcal{F}_{RE}$ ideal functionality in the $(\mathcal{F}_{crs}, s\mathcal{F}_{SAT})$ -hybrid model under static corruptions. The protocol is based on the DDH assumption, runs in a constant number of rounds, and has a communication complexity of O(n) group elements per session key.

Proof (Theorem 7). The proof of this theorem is the proof of [12, Theorem 2] verbatim. First, we observe that the multi-session version of \mathcal{F}_{RE} can be implemented by having access to the multi-session version of \mathcal{F}_{sOT} (each new session of \mathcal{F}_{RE} uses a new invocation of the protocol for \mathcal{F}_{sOT}). Then, we observe that our protocol in Section 3 implements the multi-session version of \mathcal{F}_{sOT} in the \mathcal{F}_{crs} -hybrid model. Hence, we can invoke Lemma 3, which allows us to replace \mathcal{F}_{SAT} with $s\mathcal{F}_{SAT}$, yielding a protocol for the split version of randomized equality (i.e., $s\mathcal{F}_{RE}$).

All that remains to show is that \mathcal{F}_{PAKE} can be instantiated from $s\mathcal{F}_{RE}$. Intuitively, the power of the adversary to disconnect parties in $s\mathcal{F}_{RE}$ can be mapped to TestPwD queries in \mathcal{F}_{PAKE} , as the adversary is allowed to run \mathcal{F}_{RE} with an arbitrary password by impersonating a disconnected party's peer.

Theorem 8. There exists a protocol in the $s\mathcal{F}_{RE}$ -hybrid model that instantiates \mathcal{F}_{PAKE} in the presence of subversion attacks.

Dupont et al. [22] exhibit a trivial protocol in the $s\mathcal{F}_{RE}$ -hybrid model that realizes \mathcal{F}_{PAKE} . In particular, their protocol exclusively interacts with $s\mathcal{F}_{RE}$. This fact allows to port their protocol and its related proof to our setting in a straight-forward manner, as intuitively such a protocol inherits the structure and the security properties of $s\mathcal{F}_{RE}$. We report the formal proof in Appendix C.9.

6.3 A hand-wavy performance comparison

An alternative route to obtain $\mathcal{F}_{\mathsf{PAKE}}$ consists of invoking Theorem 4 to obtain a protocol wsrUC-realizing $s\mathcal{F}_{\mathsf{RE}}$, and then applying the transformation of Theorem 8. In particular, as per Lemma 2, this protocol relies on the srUC GMW compiler of [18]. In order to establish an informal comparison with our instantiation from DDH (given by Theorem 7), we first observe that both these protocols rely on Lemma 3 to move from the authenticated setting to the unauthenticated setting. Hence, it suffices to compare the protocols in the authenticated setting. For our hand-wavy comparison, we compare round complexity and communication complexity.

Our instantiation from DDH, as per Figure 4, essentially relies on n runs of \mathcal{F}_{sOT} that share the same CRS. By our specific instantiation of \mathcal{F}_{sOT} , each party sends 1 public key and 2 SHDME encryptions (= 4

group elements) for each bit of the password. Hence, our protocol runs in 2 rounds (by batching messages for sOTs) with a communication complexity of O(n) group elements.

On the other hand, the instantiation from the srUC GMW compiler requires each party to (i) generate its random tape jointly with its peer; (ii) commit to its input; (iii) prove in zero-knowledge that each step of a semi-honest protocol realizing \mathcal{F}_{RE} was executed correctly. (i) requires 3 rounds: 1 for committing to some locally-generated randomness and 2 from the coin tossing functionality. (ii) requires 1 round. (iii) requires at least the same number of rounds of a semi-honest execution of an r-round protocol realizing \mathcal{F}_{RE} . Hence, we end up with at least 4 + r rounds. We then observe that the coin tossing functionality of [18, Section 4] relies on the sanitizable commitment functionality (presented in [18, Section 3]), which is realized by computing and forwarding bit-wise commitments (each containing 2 group elements) under the DDH assumption. Given that the input to the semi-honest instantiation of \mathcal{F}_{RE} is an n-bit password, and that the random strings used to generate the random tape have size λ , the communication complexity of the first two steps is already $O(n + \lambda)$.

We conclude that our instantiation from DDH has a better round and communication complexity even prior to the run of the compiled semi-honest instantiation of \mathcal{F}_{RE} of the protocol from GMW. We further remark that, in step (iii), the protocol from GMW requires the generation of re-randomizable NIZK arguments for each message of the protocol, hindering the efficiency further.

7 Conclusions

We presented the first subversion-resilient UC protocol for PAKE. We formalized and instantiated 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., $s\mathcal{F}_{SAT}$) in the two-tier model without assuming a PKI. Finally, we instantiated \mathcal{F}_{PAKE} by replacing, in a sanitized protocol for \mathcal{F}_{RE} , the \mathcal{F}_{SAT} assumption with $s\mathcal{F}_{SAT}$. Several interesting research questions remain open, such as fully instantiating \mathcal{F}_{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] Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021*, *Part IV*, volume 13093 of *LNCS*, pages 711–741. Springer, Cham, December 2021.
- [2] 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, Cham, December 2017.
- [3] 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, Cham, May 2020.
- [4] Paula Arnold, Sebastian Berndt, Jörn Müller-Quade, and Astrid Ottenhues. Protection against subversion corruptions via reverse firewalls in the plain universal composability framework. Cryptology ePrint Archive, Report 2023/1951, 2023.
- [5] 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 19International Conference on Applied Cryptography and Network Security, volume 11464 of LNCS, pages 465–485. Springer, Cham, June 2019.

- [6] 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.
- [7] 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, Berlin, Heidelberg, August 2005.
- [8] 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, Berlin, Heidelberg, December 2016.
- [9] 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, Berlin, Heidelberg, August 2014.
- [10] 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, Berlin, Heidelberg, December 2001.
- [11] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In 42nd FOCS, pages 136–145. IEEE Computer Society Press, October 2001.
- [12] 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, Berlin, Heidelberg, May 2012.
- [13] 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, Berlin, Heidelberg, May 2005.
- [14] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, EUROCRYPT 2003, volume 2656 of LNCS, pages 68–86. Springer, Berlin, Heidelberg, May 2003.
- [15] 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.
- [16] 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, Cham, August 2020.
- [17] 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, Cham, December 2021.
- [18] Suvradip Chakraborty, Bernardo Magri, Jesper Buus Nielsen, and Daniele Venturi. Universally composable subversion-resilient cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, EURO-CRYPT 2022, Part I, volume 13275 of LNCS, pages 272–302. Springer, Cham, May / June 2022.
- [19] 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, Berlin, Heidelberg, December 2016.

- [20] 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, Berlin, Heidelberg, March 2015.
- [21] 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, Berlin, Heidelberg, August 2016.
- [22] 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, EURO-CRYPT 2018, Part III, volume 10822 of LNCS, pages 393–424. Springer, Cham, April / May 2018.
- [23] 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, Cham, March 2018.
- [24] 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.
- [25] 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.
- [26] Adam Groce and Jonathan Katz. A new framework for password-based authenticated key exchange. Cryptology ePrint Archive, Report 2010/147, 2010.
- [27] 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, Berlin, Heidelberg, April 2015.
- [28] 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, Berlin, Heidelberg, August 2008.
- [29] Magnus Ringerud. Note on subversion-resilient key exchange. Cryptology ePrint Archive, Report 2023/749, 2023.
- [30] Lawrence Roy and Jiayu Xu. A universally composable PAKE with zero communication cost (and why it shouldn't be considered UC-secure). In Alexandra Boldyreva and Vladimir Kolesnikov, editors, PKC 2023, Part I, volume 13940 of LNCS, pages 714–743. Springer, Cham, May 2023.
- [31] 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, Berlin, Heidelberg, December 2016.
- [32] 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.
- [33] 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, Berlin, Heidelberg, August 1984.
- [34] Gustavus J. Simmons. A secure subliminal channel (?). In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 33–41. Springer, Berlin, Heidelberg, August 1986.

A Channel assumptions in the srUC Framework

In Figure 7, we compare the backbone of communication offered by \mathcal{F}_{SAT} and $s\mathcal{F}_{SAT}$. 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.

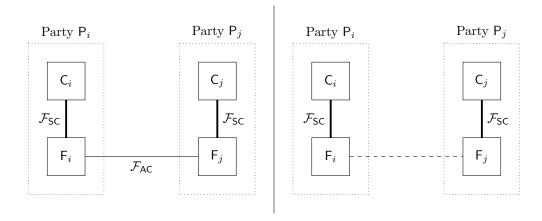


Figure 7: On the left, communication as in \mathcal{F}_{SAT} . On the right, communication as in $s\mathcal{F}_{SAT}$. \mathcal{F}_{SC} , \mathcal{F}_{AC} represent secure channels and authenticated channels, respectively.

B Basics of the UC Framework

We briefly recap the UC framework from [11]. A protocol Π consists of code for each of the parties P_1, \ldots, 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, (sid, 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 $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ 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 [11].

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 $\text{Exec}_{\mathcal{F},\mathcal{S},\mathcal{E}}$, 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:

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

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} .

C Security proofs

In this section, we report the proofs that were deferred throughout the composition.

C.1 Proof of Theorem 1

Proof (Theorem 1). Correctness and all the properties related to the original dual-mode cryptosystem are unchanged (as per [28, Theorem 5.2]). 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, $\operatorname{crs} = (g_0, g_1, h_0, h_1)$, $(pk, sk) = ((g_\sigma^r, h_\sigma^r), r)$, $\widetilde{pk}_{\mathsf{DDH,b}} = (g_b, h_b, \widetilde{pk}_0, \widetilde{pk}_1) = (g_b, h_b, g_\sigma^r, h_\sigma^r)$, and s, s', t, t' are random coins sampled by the DDHEnc algorithm.

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

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

the DDHEnc algorithm.

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

C.2 Proof of Theorem 2

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 4. The firewall $F_{\rm S}$ of the sender $C_{\rm S}$ in Figure 3 is strongly sanitizing.

Proof (Lemma 4). 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 3 with an incorruptible C_S from one that accepts the corruption \tilde{C}_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{C}_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 AlignEnc and the internal coins tossed in the computation of (y'_0, y'_1) are sampled uniformly at random and are independent from each other.
- In case (ii), if the message cannot be parsed as a tuple of ciphertexts, the message is dropped. Otherwise, the HomOp and AlignEnc algorithms produce a uniformly distributed value as y_0, y_1 are not valid ciphertexts.

Lemma 5. The firewall F_R of the receiver C_R in Figure 3 is strongly sanitizing.

Proof (Lemma 5). 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 3 with an incorruptible C_R from one that accepts the corruption \tilde{C}_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 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{C}_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 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 3 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:

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

Proof (Lemma 6). The proof of this lemma follows closely the proof of the original generic framework [28, Theorem 4.1]: 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}_{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 mode = mes; the proof for mode = 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. 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 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. (BLIND, sid, x'_0, x'_1) on S_S .
- 2. (RECEIVER, 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 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, S first computes $x_{1-b} = \hat{x}_{1-b} *^{-1} x'_{1-b}$, 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 (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, $\mathsf{Enc}(pk,b,0^n) \stackrel{s}{\approx} \mathsf{Enc}(pk,b,x_b)$. 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. 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 (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:

⁵We assume the operator supports the inverse operation. This is the case for additive blinding (\ominus), and multiplicative blinding (multiplicative inverse).

- 1. (Send, sid, x_0, x_1) on IO_S .
- 2. (BLIND, 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 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 $(pk, sk_{\sigma}) \stackrel{s}{\approx} \mathsf{KeyGen}(\mathsf{crs}, \sigma)$ by trapdoor key generation. However, crs was generated in decryption mode, whereas the protocol expects messy mode. Since $\mathsf{SetupMessy}_1 \stackrel{c}{\approx} \mathsf{SetupDec}_1$, the two experiments are only computationally indistinguishable.

Proof (Theorem 2). 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 4 and Lemma 5, the core in case (ii) can be considered Honest [18, Lemma 2]. Hence, the statement follows directly by Lemma 6. We consider Isolate corruptions as Malicious corruptions.

C.3 Proof of Theorem 3

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

Lemma 7. The protocol in Figure 4 is correct.

Proof (Lemma 7). If $w^{\rm I} = w^{\rm R}$, then $K^{\rm I} * \widetilde{K}'' = \widetilde{K}' * K^{\rm R}$ (i.e., parties output the same key):

$$\begin{split} K^{\rm I} * \tilde{K}'' &= (r_{w^{\rm I}}^{\rm I}) * (r_{w^{\rm I}}^{\rm R} * \rho_{w^{\rm I}}^{\rm R} * \rho_{w^{\rm I}}^{\rm I}) \\ &= (r_{w^{\rm I}}^{\rm I} * \rho_{w^{\rm I}}^{\rm I} * \rho_{w^{\rm I}}^{\rm R}) * r_{w^{\rm I}}^{\rm R} \\ &= (r_{w^{\rm R}}^{\rm I} * \rho_{w^{\rm R}}^{\rm R} * \rho_{w^{\rm R}}^{\rm R}) * r_{w^{\rm R}}^{\rm R} \\ &= \tilde{K}' * K^{\rm R} \end{split}$$

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. Notice however that we do not require this condition to hold whenever the adversary is able to successfully guess the password of a honest party.

Lemma 8. The firewall F_P in Figure 4, for $P \in \{I, R\}$, is strongly sanitizing.

Proof (Lemma 8). 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 4 with an incorruptible C_P from one that accepts the corruption \tilde{C}_P .

Specious Receiver. We first show that C_R is unable to leak any information. The initiator's outbound communication consists of choosing random strings and sending them through \mathcal{F}_{sOT} . Since F_R samples coins ρ_b^R 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{C}_I .

We now 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}_{sOT} , as the inputs of the (malicious) initiator may have been crafted in a way to trigger a specious corruption in C_R in an undetectable manner. In particular, the trigger should force the specious receiver to compute an adversarially-known key K. First, we note that malicious

initiator fails to guess the password, so $r_{w^R}^I \neq r_{w^I}^I$, and $\rho_{w^R}^R \neq \rho_{w^I}^R$. However, since the responder is specious, both know the random strings $(r_{w^I}^R, r_{w^R}^R)$ may be known to the malicious initiator. Moreover, the malicious initiator receives $\tilde{r}'' = r_{w^I}^R * \rho_{w^I}^R$. Since the responder's firewall is honest and the password are distinct, any trigger the malicious initiator may craft is blinded by operation * with a fresh random string that was never used before. Hence, the output key for the specious responder will be a uniformly random key.

Specious Initiator. We first argue that no input-trigger attacks are possible. The only source of input-triggers comes from the inbound values of the first batch of \mathcal{F}_{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 trigger sampled by the malicious receiver.

We now argue that C_I is unable to leak any information. The only source of leakage comes from the outbound values in the second batch of $\mathcal{F}_{\mathsf{SOT}}$. In particular, the specious initiator could sample random strings in a biased manner, so that the output of the protocol leaks information. As before, we note that the malicious receiver fails to guess the password, so $r_{w^R}^I \neq r_{w^I}^I$, and $\rho_{w^I}^I \neq \rho_{w^R}^I$. However, since the initiator is specious, it may know the (maliciously sampled) random strings $(r_{w^I}^R, r_{w^R}^R)$. Moreover, the malicious initiator receives $\tilde{r}'' = r_{w^I}^R * \rho_{w^I}^R$. Since the initiator's firewall is honest and the password are distinct, any outbound message from the specious initiator is blinded by operation * with a fresh random string that was never used before $(i.e., \rho_{w^I}^I \neq \rho_{w^R}^I)$. Hence, the outbound message will be a uniformly random string.

Remark 2. As expected, in the event the adversary successfully guesses the password, there are no longer any security guarantees against specious corruptions. In the first case of the previous proof, the malicious initiator can recompute from r'' the correct blinding factor used by the honest firewall of the receiver (as the randomness used by the specious receiver is known to the adversary). Since in this case the same blinding factor would be reused, the adversary may craft a contribution to the key that embeds the same blinding factor so that it gets cancelled out, forcing the specious core to output a fixed key and causing a trigger, breaking indistinguishability. In the second case, the specious initiator obtains r'' embedding the correct blinding factor used from its firewall, as well as some adversarially generated randomness that is hard-wired in the subverted implementation itself. With that, the specious initiator is able to recompute the blinding factor used by its firewall. From here, the specious initiator can sample a random string accounting for the blinding factor of its firewall, and leak information, breaking indistinguishability.

Lemma 9. For every adversary \mathcal{A} corrupting either party maliciously in an execution of the protocol Π as in Figure 3 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:

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

Proof (Lemma 9). Intuitively, the proof follows closely the proof of the non-sanitized protocol [12, Theorem 1], restricted to the case of static corruptions.

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

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 S can simulate the ideal functionality \mathcal{F}_{sOT} , hence it can obtain whatever the adversary A inputs on behalf of corrupted parties (honest parties are still internally simulated). More specifically, it simulates the first batch of OTs (in which I acts as the sender) as follows:

- 1. Whenever \mathcal{A} sends inputs (ρ_0^R, ρ_1^R) to \mathcal{F}_{sOT} acting as the responder's firewall F_R in the first OT, \mathcal{S} records the tuple.
- 2. Whenever \mathcal{A} sends input w to $\mathcal{F}_{\mathsf{sOT}}$ acting as a malicious initiator in the first OT, if \mathcal{S} had received a message as in (1), it records the input w, samples a random value $r' \leftarrow_{\$} \mathcal{I}$, and returns $r' * \rho_w^{\mathsf{R}}$ to \mathcal{A} . Otherwise, it does nothing.

It then simulates the second batch of OTs (in which I acts as the receiver) as follows:

- 1. Whenever \mathcal{A} sends inputs $(r_0^{\mathrm{I}}, r_1^{\mathrm{I}})$ to $\mathcal{F}_{\mathsf{sOT}}$ acting as a malicious initiator in the second OT, \mathcal{S} records the tuple.
- 2. Whenever \mathcal{A} sends inputs (ρ_0^R, ρ_1^R) to \mathcal{F}_{sOT} acting as the receiver's firewall F_R in the second OT, \mathcal{S} records the tuple.

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).

After this preliminary phase, S sends message (NewSession, sid, I, R, w) to $\mathcal{F}_{\mathsf{RE}}$. It also sends message (OK, 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, $K^{\mathsf{I}} = r_w^{\mathsf{I}} * \rho_w^{\mathsf{I}} * \rho_w^{\mathsf{R}}$, and $\widetilde{K}'' = r'$. S then sends a NewKey query containing $K = K^{\mathsf{I}} * \widetilde{K}''$ to $\mathcal{F}_{\mathsf{RE}}$ for both parties, receiving skey_I (i.e., the output of the corrupted initiator). skey_I is forwarded to \mathcal{A} .

Malicious Receiver. S initially waits for a message (sid, I, R) from \mathcal{F}_{RE} . As soon as the message is received, it inputs (OK, sid) to \mathcal{F}_{RE} , receiving a matching (WAKEUP, sid, I, R) message. From this point on, it simulates the RESPOND and NEWKEY queries 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 Honest core and a SemiHonest firewall, or (ii) a Specious core and a Honest firewall. Both firewalls are strongly sanitizing, and as shown in Lemma 8, the core in case (ii) can be considered Honest [18, Lemma 2]. Hence, the statement follows directly by Lemma 9. We consider Isolate corruptions as Malicious corruptions.

C.4 Proof of Lemma 1

Proof (Lemma 1). Assume by contradiction the theorem statement does not hold. Then, there exists an adversary \mathcal{A}' that breaks KS-EUF-CMA security of an EUF-CMA scheme with the algorithms and properties of Definition 5 with non-negligible probability. Leveraging on \mathcal{A}' , we build a reduction \mathcal{A} that breaks the EUF-CMA property of the referenced scheme with the same probability.

 \mathcal{A} acts as the challenger for KS-EUF-CMA security of \mathcal{A}' . Therefore, it has to (i) provide a verification key and randomness, and (ii) answer to signature queries. For (i), \mathcal{A} simply mauls the verification key vk received from \mathcal{C} to $\widetilde{\mathsf{vk}}$ via algorithm MaulVK using a uniformly distributed randomness ρ ; for (ii), \mathcal{A} forwards signature queries on m_i 's to \mathcal{C} , and mauls signatures σ_i to $\widetilde{\sigma}_i$ via algorithm AlignSig using the same randomness ρ used to maul vk. Finally, \mathcal{A}' will produce a forged signature σ^* on $m^* \notin \mathcal{M}$ that verifies under $\widetilde{\mathsf{vk}}$. Using algorithm UnAlignSig with randomness ρ , \mathcal{A} can translate this forgery to a forged signature σ for $m^* \notin \mathcal{M}$ that verifies under vk. This concludes the proof.

We depict pictorially the reduction in Figure 8.

C.5 Proof of Theorem 5

Proof (Theorem 5). First, we inspect the MaulVK, AlignSig, and UnAlignSig algorithms we introduced, showing that they satisfy the consistent key sanitation and alignment invertibility properties. Then, we show that the BLS signature scheme, combined with the algorithms above, achieves KS-EUF-CMA security. Finally, we inspect the IDComb algorithm we introduced to show consistent identity combinability.

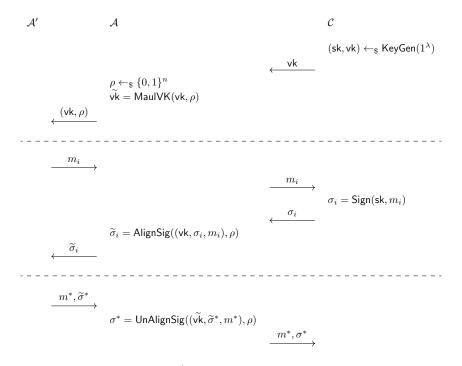


Figure 8: A reduction proving Lemma 1. \mathcal{A}' is the algorithm that breaks KS-EUF-CMA with non-negligible probability. \mathcal{A} acts as the challenger for \mathcal{A}' and as the adversary for EUF-CMA. \mathcal{C} is the challenger for EUF-CMA.

Consistent key sanitation. 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., $\mathsf{Vrfy}(\widetilde{\mathsf{vk}}, (\widetilde{\sigma}, m)) : \widehat{e}(\widetilde{\sigma}, g) \stackrel{?}{=} \widehat{e}(H(m), \widetilde{\mathsf{vk}})$) succeeds:

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

Alignment invertibility.

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

KS-EUF-CMA security. Given that the BLS signature scheme is EUF-CMA (as shown in [10]), we simply invoke Lemma 1.

Consistent identity combinability.

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

C.6 Proof of Theorem 6

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 10. The firewall F_i in Figure $\frac{4}{4}$, is strongly sanitizing.

Proof (Lemma 10). 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 5, 6 with an incorruptible C_i from one that accepts the corruption \tilde{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_i and (sid_i, 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 .
- 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 $\widetilde{\operatorname{sid}}_i$).
- σ_i 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{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 .
- \bullet 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 (P_j, \tilde{m}_i, cnt) , which is output by the honest F_i , so no triggering exists.

The only outbound message from C_i is the signature σ_j , which gets sanitized by means of algorithm 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 11. For every adversary 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 5, 6, there exists a simulator S such that, for all environments E, assuming secure channels between cores and their firewall and a KS-EUF-CMA signature scheme with consistent identity combinability, the following holds:

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

Proof (Lemma 11). The proof runs similarly to the one of Barak et al. (Proof of Theorem 11, [7]): the simulator 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 KS-EUF-CMA of the signature scheme.

Simulating 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 6). 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 10, the core in case (ii) can be considered Honest (Lemma 2 of [18]). Hence, the statement follows directly by Lemma 11. We do not consider Isolate corruptions explicitly, as they collapse to MITM attacks.

C.7 Proof of Lemma 2

Proof (Lemma 2). We follow the proof of [7] adjusting to our setting when necessary. Let $\Phi_{\mathcal{F}}$ be the protocol of [18] (i.e., the GMW compiler in the srUC framework) 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 $\pi_{\hat{\mathcal{F}}_{C\&P}}$ for realizing a multi-instance commit-and-prove functionality $\hat{\mathcal{F}}_{C\&P}$ that provides multiple independent commit-and-prove operations making (implicit) use of a single CRS functionality \mathcal{F}_{crs} , and (2) a protocol $\Pi_{\mathcal{F}}$ for realizing functionality \mathcal{F} assuming access to $\hat{\mathcal{F}}_{C\&P}$. We can express the protocol $\Phi_{\mathcal{F}}$ as $\Phi_{\mathcal{F}} = \Pi_{\mathcal{F}}^{\pi_{\mathcal{F}_{C\&P}}}$. Observe that $\pi_{\hat{\mathcal{F}}_{C\&P}}$ 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 $\hat{\mathcal{F}}_{C\&P}$ is exactly the same as running the same n instances of $\Phi_{\mathcal{F}}$ where all instances use a single joint instance of $\hat{\mathcal{F}}_{C\&P}$. Then, we can invoke the composition theorem of the srUC model and replace the the joint instance of $\hat{\mathcal{F}}_{C\&P}$ by a single instance of a protocol that realizes $\hat{\mathcal{F}}_{C\&P}$. We end up with a protocol that realizes n independent instances of n and uses a single instance of n and n and n and n and n and n are the protocol that realizes n and n and n are the protocol n and n are the protocol n

C.8 Proof of Lemma 3

Proof (Lemma 3). The first important observation is that protocols realizing regular ideal functionalities exclusively allow cores to have an input or output. Given that, the proof is essentially the same as in [7], with particular care in adapting the communication between components of parties consistently. Towards this goal, we first provide a description of the compiler of [7] adapted to our setting.

Informally, protocol $\Pi_{\mathcal{F}}$ is described as follows. Each party first runs $s\mathcal{F}_{\mathsf{SAT}}$ to obtain sid_H . Next, it runs $\pi_{\mathcal{F}}$ with $\mathsf{SID} = (\mathsf{sid}, \mathsf{sid}_H)$. We recall that, since $\pi_{\mathcal{F}}$ n-realizes \mathcal{F} , it internally runs n concurrent instances of a protocol realizing \mathcal{F} , each identified by a different sub-session ID. Each outgoing message of $\pi_{\mathcal{F}}$ is forwarded to $s\mathcal{F}_{\mathsf{SAT}}$, and each incoming message from $s\mathcal{F}_{\mathsf{SAT}}$ is forwarded to $\pi_{\mathcal{F}}$. As in [7], all participants of each

instance of $\pi_{\mathcal{F}}$ interact with a single instance of $\mathcal{F}_{\mathsf{SAT}}$, identified by SID $\mathsf{sid}||0$. We proceed more formally in what follows.

Protocol $\Pi_{\mathcal{F}}$

Let $\pi_{\mathcal{F}}$ be a protocol *n*-realizing functionality \mathcal{F} , that has access to \mathcal{G} and assumes authenticated communication (*i.e.*, $\mathcal{F}_{\mathsf{SAT}}$). Protocol $\Pi_{\mathcal{F}}$ proceeds as follows, using ideal access to \mathcal{G} and $s\mathcal{F}_{\mathsf{SAT}}$ and interacting with a set of parties, each composed of a core C_i and a firewall F_i :

- Upon receiving input (Init, sid) from party P, where P = (C, F), sid = (P, sid'), and P is a set of parties containing P, call $s\mathcal{F}_{SAT}$ with input (Init, sid||1).
- Upon receiving (INIT, $sid||1, sid_H$) from $s\mathcal{F}_{SAT}$ with destination party P, store (P, sid_H) and output (INIT, sid, sid, sid, to ITI P.
- Upon receiving (INPUT, P_i , sid, x) from core C_i of party P_i , input x to ITI ($\pi_{\mathcal{F}}$, (sid, sid_H), P_i), *i.e.*, to the ITI running $\pi_{\mathcal{F}}$ with SID (sid, sid_H) and PID P_i on C_i .
- Handle communication as follows:
 - When core C_i of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_i)$ sends message (SEND, $\mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m$) using $\mathcal{F}_{\mathsf{SAT}}$ on interface IO_i , input the tuple to $s\mathcal{F}_{\mathsf{SAT}}$ on IO_i .
 - Upon receiving (Send, sid, P_i , P_j , m) from $s\mathcal{F}_{SAT}$ on interface S_i , forward the tuple to the firewall F_i of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), P_i)$ as if coming from interface S_i of \mathcal{F}_{SAT} .
 - When firewall F_i of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_i)$ sends message (SEND, $\mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m$) using $\mathcal{F}_{\mathsf{SAT}}$ on interface S_i , input the tuple to $s\mathcal{F}_{\mathsf{SAT}}$ on S_i .
 - Upon receiving (Receive, sid, P_i , P_j , m) from $s\mathcal{F}_{SAT}$ on interface S_j of party P_j , forward the tuple to firewall F_j of ITI $(\pi_{\mathcal{F}}, (\text{sid}, \text{sid}_H), P_j)$, as if coming from \mathcal{F}_{SAT} on interface S_j .
 - When firewall F_j of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_j)$ sends message (Receive, $\mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m$) using $\mathcal{F}_{\mathsf{SAT}}$ on interface S_j , input the tuple to $s\mathcal{F}_{\mathsf{SAT}}$ on S_j .
 - Upon receiving (RECEIVE, sid, P_i , P_j , m) from $s\mathcal{F}_{SAT}$ on interface IO_j of party P_j , forward (RECEIVE, sid, P_i , P_j , m) to core C_j of ITI $(\pi_{\mathcal{F}}, (\text{sid}, \text{sid}_H), P_j)$ as if coming from \mathcal{F}_{SAT} on interface IO_j .
- When core C_i of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_i)$ outputs a value y, output y.

The goal of this proof consists in showing that $\Pi_{\mathcal{F}}$ wsrUC realizes $s\mathcal{F}$, i.e.:

$$\mathrm{Exec}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{E}}^{s\mathcal{F}_{\mathsf{SAT}}} pprox \mathrm{Exec}_{s\mathcal{F},\mathcal{S},\mathcal{E}}$$

To this extent, we build a series of intermediate experiments relating the unauthenticated execution of protocol $\Pi_{\mathcal{F}}$ to the authenticated concurrent execution.

Intermediate experiment 1. Given an adversary \mathcal{A} that interacts with a single instance of protocol $\Pi_{\mathcal{F}}$ (running on unauthenticated channels), we construct an adversary \mathcal{A}' that interacts with n instances of protocol $\pi_{\mathcal{F}}$ (running on authenticated channels). We first build an experiment $\text{Exec}_{\pi_{\mathcal{F}}, \mathcal{A}', \mathcal{E}'}^{\mathcal{F}_{\text{SAT}}}$ for the authenticated concurrent of protocol $\pi_{\mathcal{F}}$, and argue that the following holds:

$$\mathrm{Exec}^{s\mathcal{F}_{\mathsf{SAT}}}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{E}} \equiv \mathrm{Exec}^{\mathcal{F}_{\mathsf{SAT}}}_{\pi_{\mathcal{F}},\mathcal{A}',\mathcal{E}'}$$

The environment \mathcal{E}' internally runs the environment in a black-box manner \mathcal{E} , and the adversary \mathcal{A}' internally runs the adversary in a black-box manner \mathcal{A} (i.e., objects of the unauthenticated execution of protocol $\Pi_{\mathcal{F}}$), emulating $s\mathcal{F}_{\mathsf{SAT}}$ for \mathcal{E} and \mathcal{A} . We proceed to describe \mathcal{E}' and \mathcal{A}' more formally.

Environment \mathcal{E}' with black-box access to \mathcal{E} .

- Whenever \mathcal{E} wishes to input (Init, sid) to party P = (C, F), \mathcal{E}' forwards (Internal-Init, P, sid) to \mathcal{A}' .
- Whenever \mathcal{E}' receives an output (Internal-Init, sid , P, H, sid_H) from \mathcal{A}' , store (P, sid , sid_H) and forward

the tuple to \mathcal{E} as if it came from P with SID sid.

- Whenever \mathcal{E} wishes to input (INPUT, sid, x) to party P of SID sid, \mathcal{E}' checks whether there exists a stored tuple (P, sid, sid_H, H). If so, \mathcal{E}' inputs x to party P of SID (sid, sid_H) of $\pi_{\mathcal{F}}$. Otherwise, \mathcal{E}' ignores the message.
- Whenever \mathcal{E} wishes to input any other value to P, \mathcal{E}' ignores the message.
- Whenever \mathcal{E} wishes to input any value to \mathcal{A} (including any instruction for any specious core), \mathcal{E} forwards this message to \mathcal{A}' .
- Whenever \mathcal{E}' receives any message from \mathcal{A}' , it forwards the message to \mathcal{E} .
- Whenever \mathcal{E}' receives an output from party P with SID (sid, sid_H), the message is forwarded to \mathcal{E} as if it came from P with SID sid.
- Whenever \mathcal{E} halts outputting z, \mathcal{E}' also halts outputting z.

Adversary \mathcal{A}' with black-box access to \mathcal{A} .

Link initialization phase

- Whenever \mathcal{A}' receives (Internal-Init, P, sid) from \mathcal{E}' , \mathcal{A}' initializes an instance of $s\mathcal{F}_{SAT}$ with input (Init, sid||1) as if coming from party (P, sid). Messages from $s\mathcal{F}_{SAT}$ to the adversary are delivered to \mathcal{A} .
- Whenever \mathcal{A} outputs (INIT, $sid||1, P, H, sid_H$) to $s\mathcal{F}_{SAT}$, \mathcal{A}' forwards the tuple to the internal instance of $s\mathcal{F}_{SAT}$.
- Whenever the internal instance of $s\mathcal{F}_{SAT}$ outputs (INIT, $sid||1, H, sid_H$) to P, \mathcal{A}' outputs (INTERNAL-INIT, sid, P, H, sid_H) to \mathcal{E}' .

Inbound communication from $\mathcal{F}_{\mathsf{SAT}}$

- Whenever \mathcal{A}' receives a message (Send, P_i , P_j , m) from \mathcal{F}_{SAT} with destination F_i , it inputs (Send, sid | 1, P_i , P_j , m) to $s\mathcal{F}_{SAT}$ as if coming from C_i .
- Whenever \mathcal{A}' receives a message (Send, P_i , P_j , m) from \mathcal{F}_{SAT} (*i.e.*, the tuple that is leaked to the adversary), it inputs (Send, sid||1, P_i , P_j , m) to $s\mathcal{F}_{SAT}$ as if coming from F_i .
- Whenever \mathcal{A}' receives a message (RECEIVE, P_i , P_j , m) from \mathcal{F}_{SAT} with destination C_j , it inputs (RECEIVE, sid||1, P_i , P_j , m) to $s\mathcal{F}_{SAT}$ as if coming from F_j .

Outbound communication from A

- Whenever \mathcal{A} wishes to send (Send, P_i , P_j , m) to $s\mathcal{F}_{SAT}$ on IO_i , \mathcal{A}' forwards the tuple to $s\mathcal{F}_{SAT}$ on IO_i . Upon receiving (Send, P_i , P_j , m) from $s\mathcal{F}_{SAT}$ with destination (uncorrupted) F_i , \mathcal{A}' forwards the message (Send, P_i , P_j , m) to \mathcal{F}_{SAT} on IO_i .
- Whenever \mathcal{A} wishes to send (Deliver, sid||1, P_i , P_j , m) to $s\mathcal{F}_{SAT}$, \mathcal{A}' forwards the tuple to $s\mathcal{F}_{SAT}$. Whenever $s\mathcal{F}_{SAT}$ outputs (Receive, sid||1, P_i , P_j , m) with destination (uncorrupted) F_j , \mathcal{A}' sends (Deliver, sid||0, P_i , P_j , m) to \mathcal{F}_{SAT} .
- Whenever \mathcal{A} wishes to send (RECEIVE, P_i, P_j, m) to $s\mathcal{F}_{SAT}$ on S_j , \mathcal{A}' forwards the tuple to $s\mathcal{F}_{SAT}$ on S_j . Upon receiving (RECEIVE, P_i, P_j, m) from $s\mathcal{F}_{SAT}$ with destination (uncorrupted) C_j , \mathcal{A}' forwards the message (RECEIVE, P_i, P_j, m) to \mathcal{F}_{SAT} on S_j .
- \bullet Whenever \mathcal{A} wishes to deliver any other message to an uncorrupted component, the message is ignored.

Intuitively, the fact that \mathcal{A}' is receiving inbound messages from $\mathcal{F}_{\mathsf{SAT}}$ means that some honest dummy component has sent them on $\mathcal{F}_{\mathsf{SAT}}$ with destination a corrupted component. For that, \mathcal{A}' reproduces the query made by the honest component by producing a responding query to $s\mathcal{F}_{\mathsf{SAT}}$. Conversely, \mathcal{A} may want to deliver messages to uncorrupted components. For that, \mathcal{A}' first inputs them to the local instance of $s\mathcal{F}_{\mathsf{SAT}}$. If $s\mathcal{F}_{\mathsf{SAT}}$ returns a matching tuple to \mathcal{A}' , it means that it has as destination an uncorrupted component. In this case, the tuple is forwarded on $\mathcal{F}_{\mathsf{SAT}}$ on the appropriate interface.⁶.

^aNote that if F_i is corrupted, $s\mathcal{F}_{\mathsf{SAT}}$ already internally returns this tuple to \mathcal{A} .

⁶Otherwise, it means that the communication is among corrupted components and does not need to be forwarded to \mathcal{F}_{SAT} , as it is already locally emulated.

Finally we remark that, since we only restrict our attention to static corruptions, no adaptive corruption message is present. If a party is corrupted in $\Pi_{\mathcal{F}}$ (according to the corruption translation table in Table 1) before the protocol run, the party will also be corrupted in the respective run of $\pi_{\mathcal{F}}$. This implicitly includes also specious corruptions of cores, as per Table 1.

It directly follows from the construction of \mathcal{A}' and \mathcal{E}' that the view of \mathcal{E} running within \mathcal{E}' is distributed identically to the view of \mathcal{E} in the execution of $\Pi_{\mathcal{F}}$, as desired.

Intermediate experiment 2. Since $\pi_{\mathcal{F}}$ n-securely realizes \mathcal{F} , we have that for every adversary \mathcal{A}' there exists a simulator \mathcal{S}' such that, for every environment \mathcal{E}' that opens at most a single session of \mathcal{F} , the following holds:

$$\mathrm{Exec}_{\pi_{\mathcal{F}}, \mathcal{A}', \mathcal{E}'}^{\mathcal{F}_{\mathsf{SAT}}} \overset{c}{\approx} \mathrm{Exec}_{\mathcal{F}, \mathcal{S}', \mathcal{E}'}$$

The descriptions of \mathcal{A}' and \mathcal{E}' are defined above.

Intermediate experiment 3. Finally, we construct simulator \mathcal{S} from \mathcal{S}' . In particular, we prove that:

$$\text{Exec}_{\mathcal{F},\mathcal{S}',\mathcal{E}'} \stackrel{s}{\approx} \text{Exec}_{s\mathcal{F},\mathcal{S},\mathcal{E}}$$

Essentially S internally incorporates S', and mimics for S' the interaction with environment E' by using environment E. Intuitively, the goal of S consists in (i) initializing S' consistently with $s\mathcal{F}_{\mathsf{SAT}}$, (ii) forward outbound messages with destination the concurrent execution of F with SSID sid_H to $s\mathcal{F}_{\mathsf{SAT}}$, and (iii) forward inbound messages from $s\mathcal{F}_{\mathsf{SAT}}$ to corrupted parties as if coming from F. We describe S formally next.

Simulator S with black-box access to S'.

- Whenever S receives a message (Init, sid, P) from $s\mathcal{F}$, it forwards (Internal-Init, sid, P) to S' as if it came from \mathcal{E}' .
- Whenever S' wishes to send (Internal-Init, sid, P, H, sid_H) to E', S forwards (Init, sid, P, H, sid_H) to sF.
- Whenever S' wishes to send input x to the instance of F with SID (sid , sid_H), S sends (INPUT, sid , H, x) to sF (the set H associated with sid_H can be determined from the content of the respective INTERNAL-INIT message received from S').
- Upon receiving from $s\mathcal{F}$ an output v directed at party P in instance H, \mathcal{S} forwards to \mathcal{S}' an output v for party P as if coming from \mathcal{F} (in this case, P is corrupted, according to the corruption translation table in Table 1).

As for the original proof, the only difference is that $s\mathcal{F}$ might ignore certain inputs and not forward them to the respective instance of \mathcal{F} . This happens only if any of the following occurs:

- Upon receiving a message from an uncorrupted party that is not part of any H.
- Upon receiving a message (INIT, sid, P_i , H, sid_H) from the adversary that is malformed. This happens if:
 - $-P_i$ has not previously sent any (INIT, sid) message.
 - $-P_i$ is not part of set H.
 - $-s\mathcal{F}$ has previously received another pair $(\operatorname{sid}_{H'}, H')$ such that either H = H' and $H \cap H'$ contains uncorrupted parties, or H = H' but $\operatorname{sid}_H \neq \operatorname{sid}_{H'}$.

The first event never occurs, as these inputs are also ignored in the execution of \mathcal{E}' and \mathcal{A}' with (the multi-session) \mathcal{F} . In particular, \mathcal{E}' ignores all inputs that \mathcal{E} wishes to send to parties P_i that are not of a previously-recorded set H.

We analyze the second event next. By construction of \mathcal{A}' , \mathcal{A}' never outputs a malformed message (Internal-Init, sid, P, H, sid_H) to \mathcal{E}' : indeed, \mathcal{A}' stores a tuple (sid_H, H) only if $s\mathcal{F}_{SAT}$ would have done

so, which implies that only well-formed INTERNAL-INIT messages are output to \mathcal{E}' . Since for every \mathcal{A}' there exists a valid simulator \mathcal{S}' , \mathcal{S}' will output a malformed INTERNAL-INIT message with negligible probability. By construction of \mathcal{S} , the probability that \mathcal{S} sends a malformed (INIT, $\operatorname{sid}, P, H, \operatorname{sid}_H$) message to $s\mathcal{F}$ is negligible (as this occurs only if \mathcal{S}' outputs an INTERNAL-INIT that is malformed). Since the execution of \mathcal{S}' and \mathcal{E}' interacting with a multi-session \mathcal{F} is otherwise identically distributed to an execution of \mathcal{S} and \mathcal{E} interacting with $s\mathcal{F}$, the two executions are statistically indistinguishable, concluding the experiment.

Putting everything together. By putting the experiments together, we have the following, concluding the proof:

$$\operatorname{Exec}_{\Pi_{\mathcal{F},\mathcal{A},\mathcal{E}}}^{s\mathcal{F}_{\mathsf{SAT}}} \equiv \operatorname{Exec}_{\pi_{\mathcal{F},\mathcal{A}',\mathcal{E}'}}^{\mathcal{F}_{\mathsf{SAT}}} \overset{c}{\approx} \operatorname{Exec}_{\mathcal{F},\mathcal{S}',\mathcal{E}'} \overset{s}{\approx} \operatorname{Exec}_{s\mathcal{F},\mathcal{S},\mathcal{E}}$$

Remark 3. Even though we considered a protocol $\pi_{\mathcal{F}}$ n-realizing an n-party functionality \mathcal{F} , the lemma has to be cast to the 2-party setting due to the restriction we have from the instantiation of $s\mathcal{F}_{\mathsf{SAT}}$ we propose in this composition.

C.9 Proof of Theorem 8

Proof (Theorem 8). We adapt the proof of Dupont et al. [22, Appendix D] to the \mathcal{F}_{RE} and \mathcal{F}_{PAKE} we defined in Sections 4.1, 6.1. First, we define a trivial protocol⁷ Π_{PAKE} that instantiates \mathcal{F}_{PAKE} in the $s\mathcal{F}_{RE}$ -hybrid model. Then, we show a one-to-one mapping of the adversarial behaviour within the two functionalities.

Definition and simulation of Π_{PAKE} . Compared to the original proof, in Π_{PAKE} we only distinguish explicitly the initiator I from the responder R to match the description of our ideal functionalities.

Protocol Π_{PAKE}

- Upon receiving (sid, w^{P}) from party $P \in \{I, R\}$, proceed as follows:
 - Send (INIT, sid) to $s\mathcal{F}_{RE}$ on behalf of P.
 - If P = I: Send (NewSession, sid, I, R, w^{I}) to $s\mathcal{F}_{RE}$.
 - If P = R: Upon receiving (WAKEUP, sid, I, R) from $s\mathcal{F}_{RE}$, send (RESPOND, sid, R, I, w^R) to $s\mathcal{F}_{RE}$.
- Upon receiving skey_P from $s\mathcal{F}_{RE}$, output skey_P to P.

Next, we define a simulator for Π_{PAKE} similarly to Dupont et al. [22].

Simulator S_{PAKE}

Initialization

- Upon receiving (INIT, sid) from A on behalf of any corrupted party, S_{PAKE} does nothing.
- Upon receiving (INIT, sid, P_i , H, sid_H) from A, S_{PAKE} records the partition set H for party P.

${\mathcal A}$ places I and R in the same authentication set H

- If both I and R are corrupted, S_{PAKE} internally simulates the behaviour of sF_{SA} without interacting with F_{PAKE} .
- Else (i.e., at most one among I and R are corrupted), Spake proceeds as follows:
 - Upon receiving (NewSession, sid, I, R, w^{I}) from \mathcal{A} , where I is corrupted, \mathcal{S}_{PAKE} forwards the query to \mathcal{F}_{PAKE} .
 - Upon receiving (Respond, sid, R, I, w^{R}) from \mathcal{A} , where R is corrupted, \mathcal{S}_{PAKE} forwards the query to

⁷We say it is a trivial protocol, as it exclusively makes parties interact with $s\mathcal{F}_{RE}$.

 $\mathcal{F}_{\mathsf{PAKF}}$

- Upon receiving (NewKey, sid, P, K) from \mathcal{A} , \mathcal{S}_{PAKE} forwards the query to \mathcal{F}_{PAKE} .
- Upon receiving (sid, \cdot) from \mathcal{F}_{PAKE} , \mathcal{S}_{PAKE} forwards the tuple to \mathcal{A} .

$\ensuremath{\mathcal{A}}$ places I and R in different authentication sets

- If an authentication set contains exclusively corrupted parties, S_{PAKE} internally simulates the behaviour of $s\mathcal{F}_{SA}$ for that authentication set without interacting with \mathcal{F}_{PAKE} .
- Else, S_{PAKE} proceeds as follows:
 - Upon receiving (NewSession, sid, I, R, w^{I}) from \mathcal{A} , where I is corrupted, \mathcal{S}_{PAKE} forwards (TestPwd, sid, R, w_{I}) to \mathcal{F}_{PAKE} .
 - Upon receiving (RESPOND, sid, R, I, w^R) from \mathcal{A} , where R is corrupted, \mathcal{S}_{PAKE} forwards (TestPwd, sid, I, w^R) to \mathcal{F}_{PAKE} .
 - Upon receiving (NewKey, sid, P, K) from \mathcal{A} , \mathcal{S}_{PAKE} forwards the query to \mathcal{F}_{PAKE} .
 - Upon receiving (sid, ·) from \mathcal{F}_{PAKE} , \mathcal{S}_{PAKE} forwards the tuple to \mathcal{A} .

Whenever \mathcal{A} places both parties in the same authentication set, no TestPwD query can occur, as the parties were allowed to mutually authenticate and the adversary can no longer man-in-the-middle. On the other hand, if \mathcal{A} is splitting parties, an authenticated set can contain at most one honest party (as the intersection of distinct authentication sets can only contain corrupted parties). This way, the adversary is effectively performing two distinct TestPwD queries against honest I and honest R, as they are in distinct authentication and the adversary can impersonate their peer.

Mapping between $s\mathcal{F}_{RE}$ and \mathcal{F}_{PAKE} . It remains to show that the output of \mathcal{S}_{PAKE} is correctly distributed. To this extent, we enumerate all possible adversarial behaviours in $s\mathcal{F}_{RE}$ and map them to the status of records in \mathcal{F}_{PAKE} in Table 2. When the adversary "attacks" party P in $s\mathcal{F}_{RE}$, it first partitions parties in different authentication sets, and then runs the individual \mathcal{F}_{RE} on behalf of its peer by using an arbitrary password. Intuitively, the attack is successful if the arbitrary password matches with w^P . For all rows, the output of both $s\mathcal{F}_{RE}$ and \mathcal{F}_{PAKE} matches by inspecting the behaviour of the functionalities for that specific adversarial behaviour. For the sake of clarity, we discuss the meaning of some of the rows in a verbose fashion.

Row 1. Parties input $w^{\rm I}$ and $w^{\rm R}$ such that $w^{\rm I} = w^{\rm R}$, and the adversary does not MITM in $s\mathcal{F}_{\sf RE}$. This means that the parties are placed within the same authentication set. This behaviour corresponds to both records being fresh in $\mathcal{F}_{\sf PAKE}$, as no TESTPWD queries can occur. If both parties are honest (i.e., $C = \emptyset$), the output of both functionalities is a random shared key. If party R is corrupted, the adversary can set the key it outputs arbitrarily, and since it knows the password used by the honest peer I, it may set the key I outputs to an arbitrary value (as intuitively, it is able to run the protocol on behalf of R with the correct password).

Row 4. Parties input $w^{\rm I}$ and $w^{\rm R}$ and the adversary does MITM in $s\mathcal{F}_{\sf RE}$ by isolating parties and impersonating their matching peer. In particular, the adversary supplies password $\tilde{w}^{\rm R}$ on behalf of R in a run against I, and $\tilde{w}^{\rm I}$ on behalf of I in a run against R. In this row, $w^{\rm I} = \tilde{w}^{\rm R}$ and $\tilde{w}^{\rm I} \neq w^{\rm R}$. This means that the adversary successfully guessed the password of I but failed to guess the password of R. In $\mathcal{F}_{\sf PAKE}$, this behaviour corresponds to the record of I being compromised and the record of R being interrupted. Therefore, the adversary can set the key output by I.

${}_{ m S}\mathcal{F}_{ m RE}$			\mathcal{F}_{PAKE}		Output of $s\mathcal{F}_{RE}$ and \mathcal{F}_{PAKE}		
No MITM	Attack I	Attack R	Record of I	Record of R	$C = \emptyset$	$C = \{R\}$	$C = \{I, R\}$
$w^{\mathrm{I}} = w^{\mathrm{R}}$			fresh	fresh	r, r	K, K'	K, K'
$w^{\mathrm{I}} \neq w^{\mathrm{R}}$			fresh	fresh	r, s	r, K'	K, K'
	$w^{\mathrm{I}} = \tilde{w}^{\mathrm{R}}$	$\tilde{w}^{\mathrm{I}} = w^{\mathrm{R}}$	compromised	compromised	K, K'	K, K'	K, K'
	$w^{\mathrm{I}} = \tilde{w}^{\mathrm{R}}$	$\tilde{w}^{\mathrm{I}} \neq w^{\mathrm{R}}$	compromised	interrupted	K, s	K, K'	K, K'
	$w^{\mathrm{I}} = \tilde{w}^{\mathrm{R}}$		compromised	fresh	K, b	K, K'	K, K'
	$w^{\mathrm{I}} \neq \tilde{w}^{\mathrm{R}}$	$\tilde{w}^{\mathrm{I}} = w^{\mathrm{R}}$	interrupted	compromised	r, K'	r, K'	K, K'
	$w^{\mathrm{I}} \neq \tilde{w}^{\mathrm{R}}$	$\tilde{w}^{\mathrm{I}} \neq w^{\mathrm{R}}$	interrupted	interrupted	r, s	r, K'	K, K'
	$w^{\mathrm{I}} \neq \tilde{w}^{\mathrm{R}}$		interrupted	fresh	r, s	r, K'	K, K'
		$\tilde{w}^{\mathrm{I}} = w^{\mathrm{R}}$	fresh	compromised	r, K'	r, K'	K, K'
		$\tilde{w}^{\mathrm{I}} \neq w^{\mathrm{R}}$	fresh	interrupted	r, s	r, K'	K, K'

Table 2: Mapping between the adversarial behaviour in $s\mathcal{F}_{RE}$ and in \mathcal{F}_{PAKE} . C is the set of corrupted parties. r, s are randomly generated keys. K, K' are adversarially generated keys. Passwords denoted as w^{P} are inputs to the protocol. Passwords denoted as \tilde{w}^{P} are supplied by the adversary while performing MITM. The column for $C = \{I\}$ is omitted, as it is symmetrical with respect to the column for $C = \{R\}$.

D Base protocols

In this section, we report some of the descriptions of non-sanitized (i.e., base protocols) and regular ideal functionalities from prior literature.

D.1 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* ideal functionality of \mathcal{F} that interacts with an adversary \mathcal{S} and a set of parties, and behaves as follows.

Initialization

- Upon activation with input (Init, sid) from party P, parse sid = (P, sid') where P is a set of parties that includes P. Forward (Init, sid, P) to the adversary S.
- Upon receiving the message (Init, sid, P, H, sid $_H$), from \mathcal{S} : Verify that party $H \subseteq \mathcal{P}$, that the list H of party identities includes P, and that for all recorded sets H', either (i) $H \cap H'$ contains only corrupted parties and $\operatorname{sid}_H \neq \operatorname{sid}_{H'}$, or (ii) H' = H and $\operatorname{sid}_H = \operatorname{sid}_{H'}$. If any check fails, do nothing. Otherwise, record (Init, sid, sid $_H$) output the tuple 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 .

Computation

- Upon receiving an input (INPUT, sid, v) from party $P \in \mathcal{P}$, find the set H such that $P \in H$, and forward the message v from P to \mathcal{F}_H . If no such set is found, ignore the input.
- Upon receiving a message (INPUT, sid, H, P, v) from S, if \mathcal{F}_H is initialized and $P \in \mathcal{P} H$, forward v to \mathcal{F}_H as if coming from party P. Otherwise, ignore the message.
- When an instance \mathcal{F}_H generates an output v for party $P \in H$, output v to P. When the output is for a party $P \in \mathcal{P} H$ or for \mathcal{S} , send the output to \mathcal{S} .

D.2 Split Authentication

Functionality \mathcal{F}_{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 (INIT, sid), where sid = (P, sid'):

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

Upon receiving query (INIT, sid, P, H, sid $_H$), from S:

Verify that party $P \in \mathcal{P}$, 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 $\operatorname{sid}_H \neq \operatorname{sid}_{H'}$, or H' = H and $\operatorname{sid}_H = \operatorname{sid}_{H'}$. If so, output (INIT, sid , sid_H) to P and record $(H, \operatorname{sid}_H)$ if not yet recorded. Else, do nothing.

Message authentication

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

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

Upon receiving query (Deliver, sid, P, P', m) from S:

- If P' did not previously receive an (INIT, sid, 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 $(P, P', m) \in \mathcal{W}$, remove one appearance of the triple from \mathcal{W} and output (RECEIVED, sid, P, P', m) to P'. Otherwise do nothing.
- Else (i.e., P' received (INIT, sid , sid_H), and either P is corrupted or P $\notin H$), output (Received, sid , P, P', m) to P', regardless of W.

As shown in Barak et al. [7], the protocol depicted in Figure 9 instantiates \mathcal{F}_{SA} .

D.3 Oblivious Transfer

We report the generic framework of Peikert et al. [28] in Figure 10.

D.4 Randomized Equality

We depict in Figure 11 a variant of the construction of RE from OT of Canetti et al. [12] using a 1-bit password.

Figure 9: Session-authenticated channel of Barak et al. [7]. 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.

$$\begin{array}{ccc} \mathbf{Sender} \ \mathbf{S}(x_0,x_1) & \mathbf{Receiver} \ \mathbf{R}(\sigma) \\ & \mathbf{CRS} = \mathbf{crs} & \\ & (sk,pk) \leftarrow_{\$} \mathsf{KeyGen}(\sigma) \\ & \underbrace{pk} & \\ & \underbrace{(y_0,y_1)} & \\ & \underbrace{(y_0,y_1)} & \\ & \mathbf{Output} \ \mathsf{Dec}(sk,y_\sigma) \end{array}$$

Figure 10: The generic framework for \mathcal{F}_{OT} of Peikert *et al.* [28].

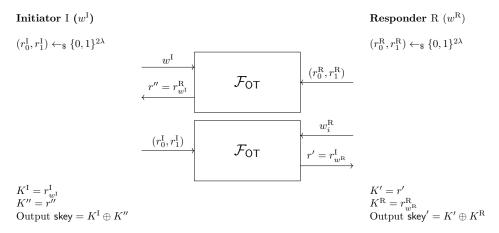


Figure 11: \mathcal{F}_{RE} from OT [12] with a 1-bit password.