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

Suvradip Chakraborty[1(✉)], Lorenzo Magliocco[2], Bernardo Magri[3,4], and Daniele Venturi[2]

[1] VISA Research, Foster City, USA
suvradip1111@gmail.com
[2] Sapienza University of Rome, Rome, Italy
[3] University of Manchester, Manchester, UK
[4] Primev, Manchester, UK

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

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

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

Keywords: PAKE · subversion resilience · universal composability

1 Introduction

Authenticated Key Exchange (AKE) allows two parties to generate a shared high-entropy secret and mutually authenticate by means of identifiers such as public keys, signatures or shared passwords. As such, AKE allows two parties

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).

K.-M. Chung and Y. Sasaki (Eds.): ASIACRYPT 2024, LNCS 15488, pp. 101–133, 2025.
https://doi.org/10.1007/978-981-96-0935-2_4

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 $\mathcal{F}_{\mathsf{SAT}}$). Following [7], we then define a weaker split-authenticated (sanitizable) variant $s\mathcal{F}_{\mathsf{SAT}}$ 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.

Sanitizing OT. Defining oblivious transfer in the presence of subversion attacks is a tricky task, as the (non-sanitized) functionality would allow a (specious) receiver to obtain exactly one of the inputs of the sender, which may act as a trigger if sampled maliciously. Similarly, it would allow a (specious) sender to sample the inputs in a leaky manner and send them over to a corrupted party. For this reason, in our sanitizable OT ideal functionality $\mathcal{F}_{\mathsf{sOT}}$ (depicted in Fig. 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.

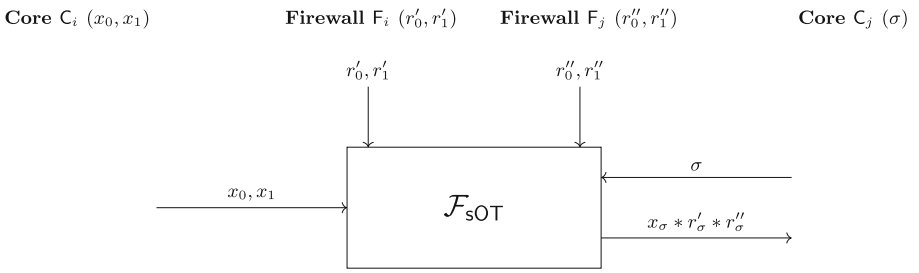

Fig. 1. Our sanitizable OT functionality $\mathcal{F}_{\mathsf{sOT}}$, with $*$ being an appropriate blinding operation for the input domain.

Here, we introduce a different technique compared to that of the seminal framework. Namely, the functionality allows firewalls to explicitly contribute to the

sanitation, and disregard their contribution whenever the overall party related to that firewall is 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 $\mathcal{F}_{\mathsf{sOT}}$, it suffices for the simulator to only forward to $\mathcal{F}_{\mathsf{sOT}}$ the malicious sender's input messages. Indeed, the notion of blinding may not even be well-defined.

In order to instantiate $\mathcal{F}_{\mathsf{sOT}}$, 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.*, $\mathsf{Enc}(m_1) * \mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 * m_2)$), (2) a procedure to maul an encryption key pk to a different encryption key $\widetilde{pk}$, and (3) a procedure to maul a ciphertext under encryption key $\widetilde{pk}$ to a ciphertext of the same message under encryption key pk. 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 $\mathcal{F}_{\mathsf{sOT}}$.

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 $\mathcal{F}_{\mathsf{sOT}}$ (*i.e.*, a protocol that realizes the the multi-session sanitizable OT functionality $\hat{\mathcal{F}}_{\mathsf{sOT}}$).

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 $\mathcal{F}_{\mathsf{OT}}$ n-times as the sender, inputting two random strings for each OT run, and n-times as the receiver, inputting the i-th bit of their password 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 $\mathcal{F}_{\mathsf{sOT}}$, designing a protocol that realizes the randomized equality ideal functionality $\mathcal{F}_{\mathsf{RE}}$ 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.

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 *unauthenticated* 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 $\widetilde{\mathsf{vk}}$, (2) a procedure to maul a signature under verification key vk into a signature under verification key $\widetilde{\mathsf{vk}}$, and (3) is equipped with a function f such that $f(\mathsf{vk}_i, \widetilde{\mathsf{vk}}_j) = f(\widetilde{\mathsf{vk}}_i, \mathsf{vk}_j)$, with $\widetilde{\mathsf{vk}}_i$ and $\widetilde{\mathsf{vk}}_j$ 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.

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

We observe that, as a corollary of the generic result of the previous paragraph, one also gets a protocol realizing $s\mathcal{F}_{\mathsf{RE}}$ 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 $\mathcal{F}_{\mathsf{RE}}$, 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.

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 [19,21], 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 [6,9,20], parameter subversion [2,3,8,23], Cliptography [5,31,32], subliminal channels [33,34] to list a few. We refer to [18,27] for further related works, such as *watchdogs* and *self-guarding*.

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 [12,26], 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 Sect. 2, we give a concise introduction to the subversion-resilient UC framework of [18]. In Sect. 3, we define and instantiate sanitizable oblivious transfer. In Sect. 4, we instantiate a subversion-resilient protocol for the randomized equality ideal functionality. In Sect. 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 Sect. 6, we combine the results of previous sections to achieve subversion-resilient PAKE. Finally, in Sect. 7, we conclude the paper with a few related open problems for further research. See Fig. 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.

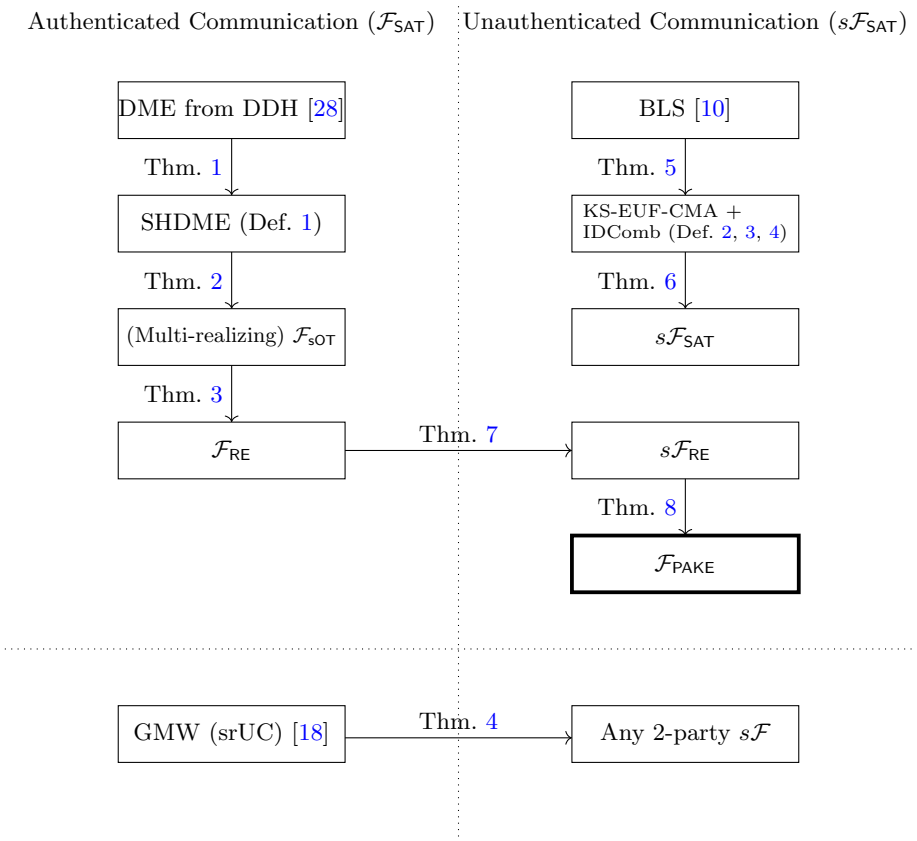

Fig. 2. A visual summary of the contributions of this paper. All the functionalities are realized in the srUC framework of [18]. DME stands for Dual-Mode Encryption. SHDME stands for Sanitizable Homomorphic DME. KS-EUF-CMA stands for Key-Sanitizable EUF-CMA. IDComb is a shorthand for Consistent Identity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

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 Honest, SemiHonest, or Malicious. Mapping the corruption possibilities for the parties $\mathsf{P}_i = (\mathsf{C}_i, \mathsf{F}_i)$ in a regular UC functionality gives rise to the following corruption translation table (Table 1):

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

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

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

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

Strong Sanitation. A firewall is strongly sanitizing if an adversary is unable to distinguish an execution of the protocol with a specious core equipped with an honest firewall from an execution of the protocol with an honest core equipped with an honest firewall. As shown in [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. We refer the reader to [18] for the formal definitions and further details.

2.3 Communication Channels

In all the protocols of [18], communication is mediated by a sanitizable ideal functionality for authenticated communication $\mathcal{F}_{\mathsf{SAT}}$, 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 $\mathcal{F}_{\mathsf{SAT}}$ 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 $\mathcal{F}_{\mathsf{crs}}$.

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

- On input $(\textsc{Send}, \mathsf{P}_i, \mathsf{P}_j, a)$ 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 $(\textsc{Send}, \mathsf{P}_i, \mathsf{P}_k, b)$ on S_i, it leaks the tuple to the adversary $\mathcal{S}$, and internally stores the tuple.
- On input $(\textsc{Deliver}, (\textsc{Send}, \mathsf{P}_i, \mathsf{P}_k, b))$ from the adversary, where the $\textsc{Send}$ tuple is stored, it outputs $(\textsc{Receive}, \mathsf{P}_i, \mathsf{P}_k, b)$ on S_k and deletes the tuple.
- On input $(\textsc{Receive}, \mathsf{P}_i, \mathsf{P}_m, c)$ on S_m, it outputs $(\textsc{Receive}, \mathsf{P}_i, \mathsf{P}_m, c)$ 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 Sects. 3 and 4, it makes little sense to instantiate PAKE by already assuming authenticated channels. In Sect. 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 Sect. 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 $\mathcal{F}_{\mathsf{sOT}}$. 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 $\mathcal{F}_{\mathsf{sOT}}$ (also denoted by $\hat{\mathcal{F}}_{\mathsf{sOT}}$).

3.1 Sanitizable OT

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

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

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

Interface IO_i:

Upon receiving a query $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ **from** C_S **on** IO_S**:**
Record $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ and forward the tuple on S_i. Ignore subsequent commands of the form $(\textsc{sender}, \mathsf{sid}, \cdot)$.

Upon receiving a query $(\textsc{receiver}, \mathsf{sid}, \sigma)$ **from** C_R **on** IO_R**:**
Check if a record $(\textsc{sender}, \mathsf{sid}, (\hat{x}_0, \hat{x}_1))$ exists. If this is the case, check the following:

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

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

Interface S_i:

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

Upon receiving a query $(\textsc{blind}, \mathsf{sid}, (x''_0, x''_1))$ **from** F_R **on** S_R**:**
If R is malicious according to the corruption translation table, do nothing. Otherwise, check the following:

* A record $(\textsc{sender}, \mathsf{sid}, (\tilde{x}_0, \tilde{x}_1))$ exists.

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

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

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 Sect. 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 refer the reader to [28, Section 3] for the formal definition and further details.

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 1 (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* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{FindMessy}, \mathsf{TrapKeyGen})$ *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* $(pk, sk) \leftarrow_\$ \mathsf{KeyGen}(\sigma)$*, for every* $\rho \in \{0,1\}^n$*,* $\mathsf{MaulPK}(pk, \rho)$ *outputs a new encryption key* $\widetilde{pk}$ *with the following property. For every* $\tilde{c} \leftarrow_\$ \mathsf{Enc}(\widetilde{pk}, \sigma, m)$*, with* $i \in \{0,1\}$ *and* $m \in \{0,1\}^n$*,* $\mathsf{AlignEnc}(c, \rho)$ *produces a new ciphertext* c *under public key* pk*, i.e.,* $\mathsf{AlignEnc}(\tilde{c}, \rho) = c$*, where* $c = \mathsf{Enc}(pk, \sigma, m)$*.*

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.

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.
- $\mathsf{KeyGen}(\sigma) = ((g_\sigma^r, h_\sigma^r), r) = ((pk_1, pk_2), sk) = (pk, sk)$.
- $\mathsf{Enc}(pk, m, b) = (g_b^s h_b^t, pk_1^s pk_2^t m) = (c_1, c_2)$.
- $\mathsf{Dec}(sk, c) = c_2 / c_1^r$.

The DDH cryptosystem is compatible with all the additional interfaces we introduced in Definition 1, and we define algorithms matching the newly introduced properties in a straight-forward manner:

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

Theorem 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 the full version.

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 $\sigma = b$.

Sanitizing the Framework. 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 Fig. 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.

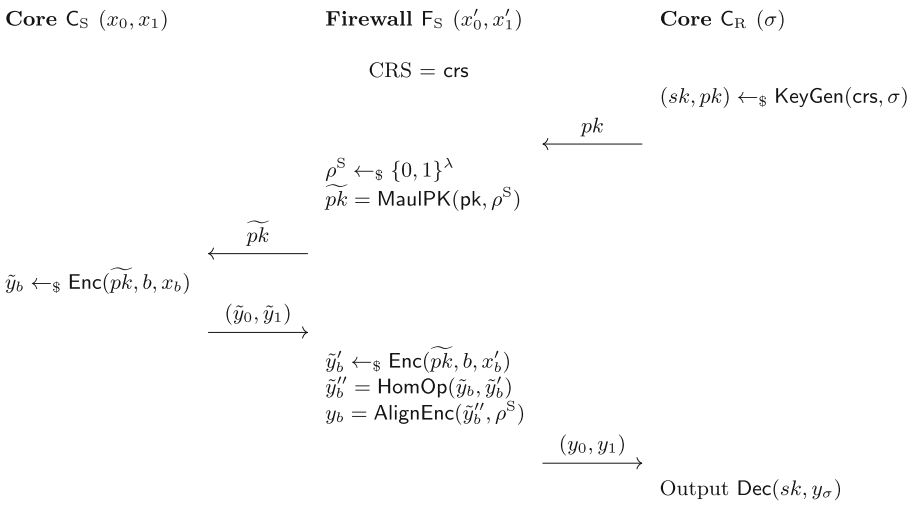

Fig. 3. A sanitation of the generic framework of Peikert *et al.* [28], realizing $\mathcal{F}_{\mathsf{sOT}}$. The receiver's firewall is omitted, as it runs the same code as F_S.

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

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 the full version.

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

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 $\mathcal{F}_{\mathsf{COM}}$ allows a committer to commit to a single value; to produce another commitment a new and independent instance of (the protocol realizing) $\mathcal{F}_{\mathsf{COM}}$ must be spawned with a brand new setup. In contrast, the multi-session functionality $\mathcal{F}_{\mathsf{MCOM}}$ allows a committer to perform poly-many commitments using the same setup. Hence, using multiple instances of $\mathcal{F}_{\mathsf{MCOM}}$ has the same effect as using a single instance of $\mathcal{F}_{\mathsf{MCOM}}$.

Moving to our case, we note that the generic framework of [28] actually realizes the multi-session version of $\mathcal{F}_{\mathsf{OT}}$ (also denoted as $\hat{\mathcal{F}}_{\mathsf{OT}}$). Given that our protocol in Fig. 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 $\mathcal{F}_{\mathsf{RE}}$ that relies on authenticated channels (i.e., $\mathcal{F}_{\mathsf{SAT}}$) and $\mathcal{F}_{\mathsf{sOT}}$, following the construction of Canetti *et al.* [12].

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

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

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

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

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

Upon receiving a query $(\textsc{ok}, \mathsf{sid})$ **from** $\mathcal{S}$**:**
Send a message $(\textsc{wakeup}, \mathsf{sid}, \mathrm{I}, \mathrm{R})$ to R. Ignore all future $(\textsc{ok})$ messages.

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

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

In both cases, ignore subsequent inputs from R.

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

- If any of the following conditions hold, output (sid, K) to party P:
  - P is corrupted.

– $w^{\mathrm{I}} = w^{\mathrm{R}}$, and the peer of P is corrupted.
- Otherwise, output $(\mathsf{sid}, \mathsf{skey}_{\mathsf{P}})$ to party P.

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

4.2 Randomized Equality from OT

We sanitize the RE from OT protocol of [12, Section 2.2] by using $\mathcal{F}_{\mathsf{sOT}}$, 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 $\mathcal{F}_{\mathsf{sOT}}$. For ease of exposition, we depict the protocol in Fig. 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 $\hat{\mathcal{F}}_{\mathsf{sOT}}$, 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 Fig. 4 wsrUC-realizes the $\mathcal{F}_{\mathsf{RE}}$ ideal functionality in the $(\mathcal{F}_{\mathsf{sOT}}, \mathcal{F}_{\mathsf{SAT}})$-hybrid model under static corruptions.*

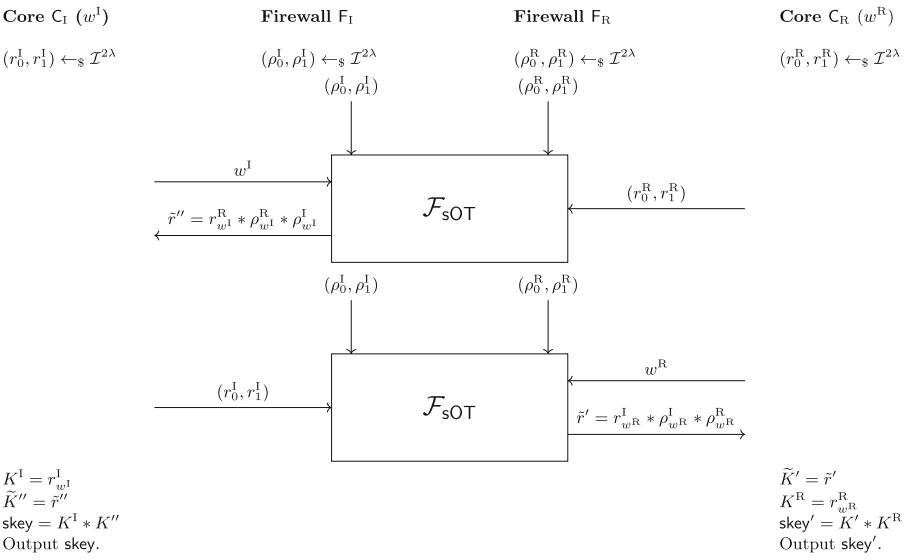

Fig. 4. A sanitizing protocol for $\mathcal{F}_{\mathsf{RE}}$ from sanitizable OT with a 1-bit password.

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 the full version.

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-formed[1] regular 2-party[2] ideal functionality $\mathcal{F}$, there exists a 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.

[1] The "well-formed" property is to rule out unrealistic functionalities as explained in [7,15].

[2] We 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.

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].

Description of $s\mathcal{F}_{\mathsf{SAT}}$. 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 $(\textsc{Init}, \mathsf{sid})$ from party P: Parse $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$ where $\mathcal{P}$ is a set of parties that includes P. Forward $(\textsc{Init}, \mathsf{sid}, \mathsf{P})$ to the adversary $\mathcal{S}$.
- Upon receiving the message $(\textsc{Init}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$, 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 $\mathsf{sid}_H \neq \mathsf{sid}_{H'}$, or (ii) $H' = H$ and $\mathsf{sid}_H = \mathsf{sid}_{H'}$. If any of the check fails, do nothing. Otherwise, output $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ to P and record (H, sid_H) if not yet recorded.

Message Authentication

- Upon receiving the message $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m)$ on IO_i where $\mathsf{P}_j \in \mathcal{P}$: Output the tuple on S_i.
- Upon receiving the message $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m})$ 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 $(\textsc{Deliver}, (\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m}))$ from $\mathcal{S}$:
  - If P_j did not previously receive an $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ 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 $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m}) \in \mathcal{W}$, remove one appearance of the tuple from $\mathcal{W}$ and output $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m})$ on S_j. Otherwise, do nothing.
  - Else (i.e., P_j received $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$, and either P_i is corrupted or $\mathsf{P}_i \notin H$), output $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m})$ on S_j, regardless of $\mathcal{W}$.
- Upon receiving the message $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \hat{m})$ 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 $\mathcal{F}_{\mathsf{SAT}}$, cores and their respective firewall are allowed to freely communicate through secure channels. This is achieved by means of Send messages (from a core to its firewall), and Receive messages (from a firewall to its core). In principle, a firewall may send back any message to its core, even if it was not related to any Deliver message from the adversary.

Key-Sanitizable Signature Schemes. In the construction of $\mathcal{F}_{\mathsf{SA}}$ 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 2, introducing a matching security notion in Definition 3 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 2 (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* $(\mathsf{vk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KeyGen}(1^{\lambda})$*, for every* $\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{sk}, m)$ *with* $m \in \{0,1\}^n$*,* $\mathsf{Vrfy}(\mathsf{vk}, (m, \sigma)) = 1$*.*
2. ***Consistent key sanitation:*** *For every* $(\mathsf{vk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KeyGen}(1^{\lambda})$*, for every* $\rho \in \{0,1\}^n$*,* $\mathsf{MaulVK}(\mathsf{vk}, \rho)$ *outputs a new verification key* $\widetilde{\mathsf{vk}}$ *with the following property. For every* $\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{sk}, m)$ *with* $m \in \{0,1\}^n$*,* $\mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho)$ *produces an accepting signature* $\widetilde{\sigma}$ *for message* m *verifiable by verification key* $\widetilde{\mathsf{vk}}$*, i.e.,* $\mathsf{Vrfy}(\widetilde{\mathsf{vk}}, \mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho)) = 1$*, where* $\widetilde{\mathsf{vk}} = \mathsf{MaulVK}(\mathsf{vk}, \rho)$ *and* $\sigma = \mathsf{Sign}(\mathsf{sk}, m)$*.*
3. ***Alignment invertibility:*** *For every* $(\mathsf{vk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KeyGen}(1^{\lambda})$*, for every* $\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{sk}, m)$ *with* $m \in \{0,1\}^n$*, for every* $\rho \in \{0,1\}^n$*, for every* $\widetilde{\mathsf{vk}} = \mathsf{MaulVK}(\mathsf{vk}, \rho)$*, for every* $\widetilde{\sigma} = \mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho)$*, the algorithm* UnAlignSig *returns the original signature* σ*, i.e.,* $\mathsf{UnAlignSig}((\widetilde{\mathsf{vk}}, \widetilde{\sigma}, m), \rho) = \sigma$

Definition 3 (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* $\mathcal{A}$ *winning the following game is negligible:*

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

Lemma 1. *Any EUF-CMA signature scheme that supports algorithms* MaulVK, AlignSig*, and* UnAlignSig*, as defined in Definition 2, is also KS-EUF-CMA.*

The proof consists of a black-box reduction to EUF-CMA, and is deferred to the the full version.

Combining Verification Keys. Looking ahead, the link initialization phase of the protocol realizing $s\mathcal{F}_{\mathsf{SAT}}$ relies on the determination of session IDs via (identifying) verification keys of parties, which get sanitized by firewalls in different directions. For instance, in the 2-party setting, core C_i has access to vk_i and $\widetilde{\mathsf{vk}}_j$, and core C_j has access to $\widetilde{\mathsf{vk}}_i$ and vk_j, with $\widetilde{\mathsf{vk}}_i$, $\widetilde{\mathsf{vk}}_j$ being appropriate sanitations of vk_i, vk_j using the same randomness 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 4 (Consistent identity combinability). *A key-sanitizable signature scheme has consistent identity combinability if it supports an algorithm* IDComb *with the following property:*

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

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

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

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:

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

Theorem 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 the full version.

Realizing $s\mathcal{F}_{\mathsf{SAT}}$. 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 Fig. 5, and the message authentication phase in Fig. 6. A verbose description of the protocol can be found in the full version.

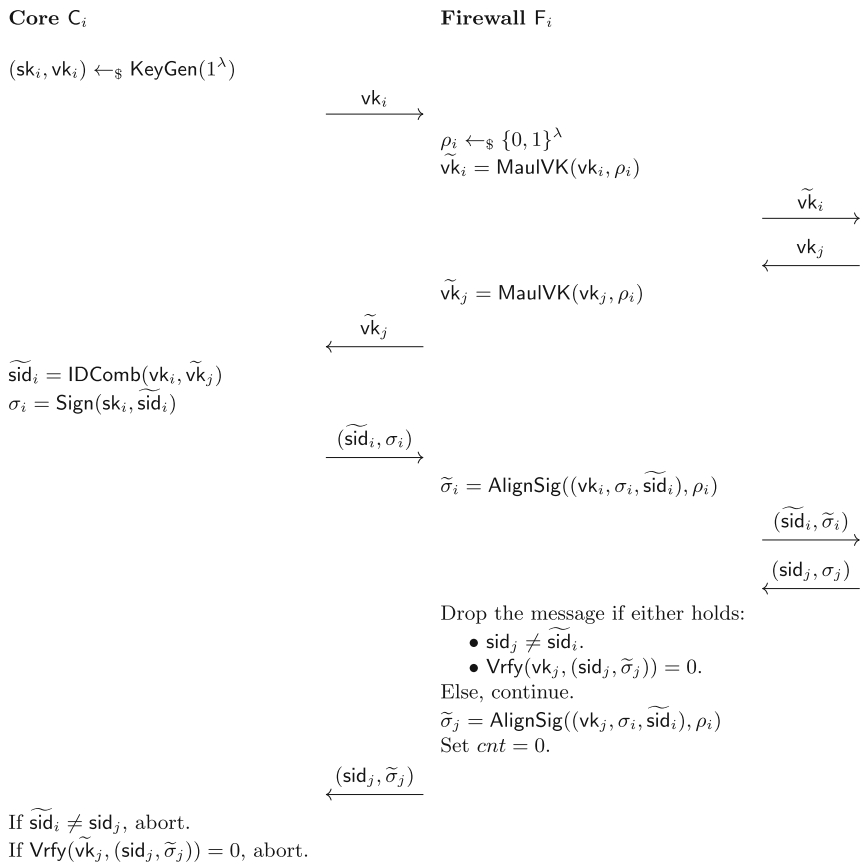

Fig. 5. Diagram of the protocol implementing the link initialization phase of $s\mathcal{F}_{\mathsf{SAT}}$.

Theorem 6. *The protocol depicted in Figs. 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 the full version.

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 $\mathcal{F}_{\mathsf{crs}}$.

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 the full version.

5.3 Realizing Generic Split Functionalities

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.

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*

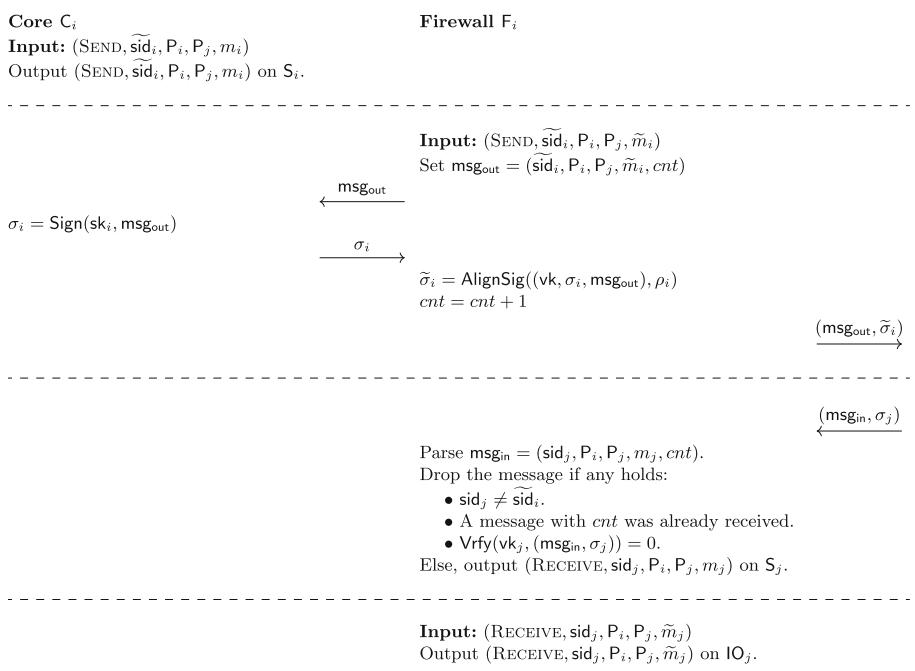

Fig. 6. Diagram of the protocol implementing the message authentication $s\mathcal{F}_{\mathsf{SAT}}$, split in each of the interfaces.

authenticated communication (i.e., $\mathcal{F}_{\mathsf{SAT}}$) 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 $s\mathcal{F}_{\mathsf{SAT}}$ 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 the full version.

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 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 $\mathcal{F}_{\mathsf{RE}}$ 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 $\mathcal{F}_{\mathsf{PAKE}}$, highlighting its differences with respect to $\mathcal{F}_{\mathsf{RE}}$. Then, similarly to [12], we argue that our protocol in Sect. 4 can be compiled in a protocol for $s\mathcal{F}_{\mathsf{RE}}$ by invoking a result of Sect. 5. Finally, we show that $s\mathcal{F}_{\mathsf{RE}}$ is sufficient to trivially realize $\mathcal{F}_{\mathsf{PAKE}}$. We conclude the section by highlighting that it is also possible to obtain a protocol for $s\mathcal{F}_{\mathsf{RE}}$ 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 Sect. 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 $\mathcal{F}_{\mathsf{PAKE}}$ is parameterized by a security parameter λ. an initiator I, a responder R, and the adversary $\mathcal{S}$ via the following queries:

Upon receiving a query $(\textsc{NewSession}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, w^{\mathrm{I}})$ **from** I**:**
Record $(\mathrm{I}, \mathrm{R}, w^{\mathrm{I}})$, mark it as fresh, and leak $(\mathsf{sid}, \mathrm{I}, \mathrm{R})$ to $\mathcal{S}$. Ignore all future messages from I.

Upon receiving a query $(\textsc{ok}, \mathsf{sid})$ **from** $\mathcal{S}$**:**
Send a message $(\textsc{wakeup}, \mathsf{sid}, \mathrm{I}, \mathrm{R})$ to R. Ignore all future $(\textsc{ok})$ messages.

Upon receiving a query $(\textsc{Respond}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, w^{\mathrm{R}})$ **from** R**:**
Record $(\mathrm{R}, \mathrm{I}, w^{\mathrm{R}})$ and mark it as fresh.

Upon receiving a query $(\textsc{TestPwd}, \mathsf{sid}, \mathsf{P}, w')$ **from the adversary** $\mathcal{S}$**:**
If $\mathsf{P} \in \{\mathrm{I}, \mathrm{R}\}$ and there exists a record of the form $(\mathsf{P}, \cdot, w)$ which is fresh, then:

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

Upon receiving a query $(\textsc{NewKey}, \mathsf{sid}, \mathsf{P}_i, K)$ **from** $\mathcal{S}$**, where** $|K| = \lambda$**:**
If $\mathsf{P}_i \in \{\mathrm{I}, \mathrm{R}\}$ and there is a record of the form $(\mathsf{P}_i, \mathsf{P}_j, w_i)$ 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 $(\mathsf{P}_j, \mathsf{P}_i, w_j)$ with $w_i = w_j$, and P_j is corrupted.
  - This record is compromised.
- If this record is fresh, both parties are honest, and there exists a record $(\mathsf{P}_j, \mathsf{P}_i, w_j)$ with $w_j = w_i$, choose $\mathsf{skey} \leftarrow_{\$} \{0,1\}^{\lambda}$. Output skey to P_i, and append skey to the record $(\mathsf{P}_i, \mathsf{P}_j, w_i)$.
- If this record is fresh, both parties are honest, and there exists a record $(\mathsf{P}_j, \mathsf{P}_i, w_j, \mathsf{skey})$ with $w_j = w_i$, output skey to P_i.
- If none of the above rules apply, choose $\mathsf{skey}' \leftarrow_{\$} \{0,1\}^{\lambda}$ and output it to party P_i.

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

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

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

Theorem 7. *There exists a protocol that wsrUC-realizes the $s\mathcal{F}_{\mathsf{RE}}$ ideal functionality in the $(\mathcal{F}_{\mathsf{crs}}, s\mathcal{F}_{\mathsf{SAT}})$-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 $\mathcal{F}_{\mathsf{RE}}$ can be implemented by having access to the multi-session version of $\mathcal{F}_{\mathsf{sOT}}$ (each new session of $\mathcal{F}_{\mathsf{RE}}$ uses a new invocation of the protocol for $\mathcal{F}_{\mathsf{sOT}}$). Then, we observe that our protocol in Sect. 3 implements the multi-session version of $\mathcal{F}_{\mathsf{sOT}}$ in the $\mathcal{F}_{\mathsf{crs}}$-hybrid model. Hence, we can invoke Lemma 3, which allows us to replace $\mathcal{F}_{\mathsf{SAT}}$ with $s\mathcal{F}_{\mathsf{SAT}}$, yielding a protocol for the split version of randomized equality (*i.e.*, $s\mathcal{F}_{\mathsf{RE}}$).

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

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

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

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 Fig. 4, essentially relies on n runs of $\mathcal{F}_{\mathsf{sOT}}$ that share the same CRS. By our specific instantiation of $\mathcal{F}_{\mathsf{sOT}}$, 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 $\mathcal{F}_{\mathsf{RE}}$ 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 $\mathcal{F}_{\mathsf{RE}}$. 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 $\mathcal{F}_{\mathsf{RE}}$ 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 $\mathcal{F}_{\mathsf{RE}}$ 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.*, $s\mathcal{F}_{\mathsf{SAT}}$) in the two-tier model without assuming a PKI. Finally, we instantiated $\mathcal{F}_{\mathsf{PAKE}}$ by replacing, in a sanitized protocol for $\mathcal{F}_{\mathsf{RE}}$, the $\mathcal{F}_{\mathsf{SAT}}$ assumption with $s\mathcal{F}_{\mathsf{SAT}}$. Several interesting research questions remain open, such as fully instantiating $\mathcal{F}_{\mathsf{SAT}}$ in the two-tier model, expanding the notion of split functionalities in the srUC model to the n-party setting, extending the framework to adaptive corruptions, weakening

trusted setups to be subvertable, and achieving explicit mutual authentication for randomized equality and PAKE.

References

1. 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, *EUROCRYPT 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, *EUROCRYPT 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.