Round-Optimal Subversion-Resilient UC PAKE from Malleable Trapdoor Smooth Projective Hash Functions

Behzad Abdolmaleki[1], Suvradip Chakraborty[2], Shahram Khazaei[3], Lorenzo Magliocco[4], Nahid Roustaeifar[1], Behzad Vahdani[3], and Daniele Venturi[4]

[1] University of Sheffield
{behzad.abdolmaleki,nroustaeifar}@sheffield.ac.uk
[2] VISA Research, USA
suvchakr@visa.com
[3] Sharif University of Technology
{shahram.khazaei,behzadvahdani1992}@gmail.com
[4] Sapienza University of Rome
{magliocco,venturi}@di.uniroma1.it

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 paper, we revisit the question of constructing PAKE protocols 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). The latter goal was recently achieved by Chakraborty, Magliocco, Magri and Venturi (ASIACRYPT 2024), based on sanitation of oblivious transfer protocols and dual-mode cryptosystems via cryptographic reverse firewalls (Mironov and Stephens-Davidowitz, EUROCRYPT 2015). Our contributions are as follows:

- We introduce so-called *malleable* trapdoor smooth projective hash functions (M-TSPHF), as an enhancement of trapdoor smooth projective hash functions (Benhamouda *et al.*, CRYPTO 2013). Our extension incorporates new properties including key malleability and element rerandomizability.
- We give a generic construction of subversion-resilient UC PAKE based on M-TSPHF and other standard cryptographic primitives. As we demonstrate, our PAKE protocol can be instantiated efficiently yielding an improved round and communication complexity with respect to the previous protocol of Chakraborty *et al.* In particular, our PAKE protocol achieves round optimality, as it concludes in a single round.

Keywords: password-authenticated key exchange; smooth projective hash functions; universal composability; subversion.

1 Introduction

Traditional cryptographic security models operate under the assumption that honest parties can fully trust the devices executing their protocols. However,

this assumption often fails in real-world scenarios. Many practical attacks exploit weaknesses in implementations rather than treating cryptographic algorithms as abstract black boxes. Moreover, users are often constrained to using hardware and software developed by external entities, such as corporations with specialized expertise or products mandated by standardization bodies. The shocking revelations from Edward Snowden [6] exposed the reality that an honest party's machine can be unknowingly compromised, whether through hardware backdoors, tampered components, or software-based attacks such as algorithm substitution attacks, implementation bugs or misconfigured protocols. This reignited the field of subversion-resilient cryptography, initiated by Young and Yung in 1996 [35,36], which aims at providing countermeasures against adversaries that manipulate cryptographic implementations to covertly leak sensitive information. As concrete examples, one can design solutions against algorithm substitution attacks [8,5,22] or parameter subversion attacks [7,2,24,4,3,1].

For this work, we focus on cryptographic reverse firewalls, originally introduced by Mironov and Stephens-Davidowitz [34] as yet another possible defense against subversion attacks. Unlike traditional firewalls that filter incoming and outgoing network traffic, a reverse firewall (RF) operates at the level of cryptographic computations, acting as an intermediary between a potentially compromised device and the outside world. Its primary function is to sanitize outgoing and incoming messages of the party they are attached to, thus effectively eliminating potential side channels. This ensures that even if an adversary compromises an implementation, the core security properties of the protocol—such as privacy and correctness—remain intact. Starting with the work of [34], most of the follow up works [5,23,21,25,15,16,17] were limited to standalone security, where each protocol operates in isolation. However, Chakraborty *et al.* [19] recently extended the setting of RFs to the universal composability (UC) framework, which ensures that once a protocol is proven secure, its subversion resilience remains intact even when arbitrarily composed with other protocols.

The work of [19] presented a construction of a maliciously secure MPC protocol within the subversion-resilient UC (srUC) framework. Specifically, they provided a sanitized version of the classical GMW compiler [28], which yields a completeness theorem for UC subversion-resilient MPC. However, this remains a feasibility result and is not practical for real-world deployment. The work of [18] was the first to construct a concrete functionality—password-authenticated key exchange (PAKE)—within the srUC framework, without relying on generic transformations from existing RF-based MPC protocols. A PAKE protocol enables two parties to derive a high-entropy secret from a shared low-entropy password. Its security ensures that an adversary cannot break the protocol with a success probability better than directly guessing the password during the online phase.[1] The protocol of [18] is constructed by sanitizing the UC PAKE-from-OT of Canetti *et al.* [11], and requires (i) setting up a sanitizable split-authenticated communication channel, and (ii) executing two subversion-resilient oblivious transfers (one per party) for each bit of the password. Consequently, its commu-

[1] The transcript of a PAKE protocol should not enable offline dictionary attacks.

nication complexity scales linearly with the length of the password, while having a sub-optimal round complexity. As a result, the protocol remains inefficient for practical deployment and is primarily of theoretical interest.

1.1 Our results

Our primary contribution is the design of the first *concretely efficient* UC PAKE protocol that remains secure under subversion attacks. Unlike [18], which requires four rounds plus additional internal interactions between cores and their firewall, our protocol achieves an efficient *single*-round srUC PAKE. Furthermore, in terms of communication complexity, our protocol transmits only a constant number of group elements in the security parameter λ, whereas [18] incurs an overhead of $O(\lambda)$ group elements for a λ-bit password. Similarly to prior works on subversion UC [19,18], we restrict our focus to so-called *specious* subversion.[2] In this model, a subverted core appears indistinguishable from an honest core to any efficient test, yet it may covertly leak private information through subliminal channels.

The starting point of our protocol is the construction of UC PAKE from trapdoor smooth projective hash functions (TSPHF), introduced in [9]. However, to ensure security within the srUC framework, we significantly modify the original building blocks to allow the RFs to properly sanitize messages. To this end, we introduce a novel concept called *malleable* TSPHF, which enables RFs to rerandomize both the projection keys and the messages of the underlying TSPHF. Additionally, we demonstrate how to instantiate a malleable TSPHF for the language of ElGamal ciphertexts, under the DDH assumption in groups with bilinear pairings. The protocol of [9] constructs a UC PAKE scheme by combining a TSPHF with a CCA-secure public-key encryption (PKE) scheme. However, in the srUC framework, the RF must sanitize the outbound ciphertexts generated by the parties, which conflicts with the inherent non-malleability of CCA-secure PKE. To address this, our protocol replaces the CCA-secure PKE with a rerandomizable CPA-secure PKE scheme, combined with a controlled-malleable non-interactive zero-knowledge (NIZK) argument system (further discussed in the technical overview). This controlled malleability allows the RF to sanitize ciphertexts while ensuring that adversaries cannot compromise the security of the protocol.

1.2 Technical overview

Below we present the main technical ideas behind our definition of M-TSPHFs and the instantiation of our srUC PAKE protocol.

[2] As noted in [19], arbitrary subversion settings lead to trivial impossibility results. Moreover, the notion of specious corruption effectively captures many practical subversion attacks, as a subverter would typically seek to remain covert while carrying out an attack.

Smooth projective hash functions. Let R be an NP relation and L be the corresponding language. An SPHF for R is a hash function that has a hashing-key hk and a projection-key hp. Using hk, one can compute the value of the hash on arbitrary inputs, but with hp the computation of the hash is only possible over L and only by having access to the witness. In particular, by exclusively having access to hp, the adversary learns nothing about the value of the hash for inputs not in L. This property is called smoothness and admits computational and information-theoretic variants.

SPHFs for the language of ciphertexts of CPA-secure and CCA-secure encryption schemes have been of special interest, primarily due to their use in the construction of efficient PAKE protocols. Within this work, we focus on CPA-security, and in particular on SPHFs for the ElGamal encryption scheme. To be concrete, we present this primitive using the language of ElGamal ciphertexts, which includes tuples of the form $(g_1^w, \mathsf{msg} \cdot h_1^w)$. The classic SPHF for this language uses a random $(\alpha_1, \alpha_2) \in \mathbb{Z}_p$ as its hashing key and $\gamma = g_1^{\alpha_1} h_1^{\alpha_2}$ as its projection key. For a ciphertext $(\mathsf{ct}_1, \mathsf{ct}_2)$, the evaluation of the hash with hk is computed as $\mathsf{ct}_1^{\alpha_1}(\mathsf{ct}_2/\mathsf{msg})^{\alpha_2}$, while the evaluation with hp and w is computed as γ^w. Note that the computation of the hash value using hk requires msg, which is generally modeled as an auxiliary input in the context of SPHFs.

Trapdoor smooth projective hash function. A variant of SPHF we consider in this paper is trapdoor SPHF [9] (**T**SPHF for short). In addition to the two hash evaluation methods described above, in a trapdoor SPHF one can compute hashes by using the projection key and a trapdoor. These hashes are consistent with the evaluation of the algorithm that uses the hashing key even if the evaluated element is not in the language. This property will be useful in proving the UC-security of the SPHF-based PAKE protocol.

The authors of [9] introduce a generic method for constructing TSPHFs using groups with bilinear pairings. Applying their method to the above SPHF for ElGamal would give the following construction, in which g_2 is the generator for the second group and e is the pairing map: The hashing key is the same as before; the projection key consists of $(\gamma, \chi_1 = g_2^{\tau\alpha_1}, \chi_2 = g_2^{\tau\alpha_2})$ where τ is the trapdoor. The hash value can be computed as follows:

$$e(\mathsf{ct}_1^{\alpha_1}(\mathsf{ct}_2/\mathsf{msg})^{\alpha_2}, g_2) = e(\gamma^w, g_2) = (e(\mathsf{ct}_1, \chi_1)e(\mathsf{ct}_2/\mathsf{msg}, \chi_2))^{\tau^{-1}}.$$

Malleable smooth projective hash function. Malleable SPHFs (**M**SPHFs), as defined in [21], enhance SPHFs with algorithms that allow rerandomizing statements (elements of the language) and projection keys. More specifically, MSPHFs introduce:

1. Algorithms that maul (or rerandomize) the projection keys (or elements of the language).
2. Algorithms that update the (unsanitized) hash value to match the new mauled projection key (or element of the language).

In the original treatment of [21], this primitive was defined with cryptographic reverse firewalls in mind. Concretely, a RF could blind a projection key (resp. an element) sent by a core with some randomness (using the first class of algorithms), and later reuse the same randomness to forward a quantity that may be used to update the (unsanitized) hash value in a consistent manner.

In all the applications of [21], the RF would act in a *transparent* fashion: namely, both classes of algorithms are *exclusively* run by firewalls. For that, the syntax of these algorithms is defined without having access to the auxiliary input of SPHFs, which is implicitly assumed to be private to the core running the functional component of the protocol (e.g., a password of a PAKE protocol). Unfortunately, this restriction makes the generic construction of [21] inapplicable to languages that mandate using the auxiliary input msg in the evaluation of hashes.

To be more concrete, we highlight why this restriction is problematic in the context of the language of ElGamal ciphertexts by considering projection-key rerandomization (element rerandomization is trivial). A natural attempt to rerandomize the projection key would consist of the following: Given the projection key $\gamma = g_1^{\alpha_1} h_1^{\alpha_2}$, sample random elements $(\widetilde{\alpha_1}, \widetilde{\alpha_2})$ and rerandomize the projection key into $g_1^{\widetilde{\alpha_1}} h_1^{\widetilde{\alpha_2}} \gamma = g_1^{\alpha_1 + \widetilde{\alpha_1}} h_1^{\alpha_2 + \widetilde{\alpha_2}}$. In order to update the hash value $hv = \mathsf{ct}_1^{\alpha_1} (\mathsf{ct}_2/\mathsf{msg})^{\alpha_2} = \gamma^w$ to match the rerandomization of the projection key, one can simply multiply hv by $\widetilde{hv} = \mathsf{ct}_1^{\widetilde{\alpha_1}} (\mathsf{ct}_2/\mathsf{msg})^{\widetilde{\alpha_2}}$. However, the rerandomization algorithm is not allowed to use msg due to the aforementioned restriction on the syntax. The alternative evaluation of $\widetilde{hv}$ as $(g_1^{\widetilde{\alpha_1}} h_1^{\widetilde{\alpha_2}})^w$ is also unsatisfactory, given that the rerandomization algorithm is not allowed to use the witness.

To solve this problem, we propose two approaches. In the main body, we tweak the syntax of [21] by providing the rerandomization algorithms with access to msg. This allows us to use the aforementioned (efficient) construction of MSPHF for the language of ElGamal ciphertexts while additionally enabling the design of more efficient malleable trapdoor SPHFs. Looking ahead, this solution is compatible with our srUC PAKE, given that the srUC framework allows firewalls to return their randomness back to their core (who knows the message msg, which in the case of PAKE is the password). In Appendix D, we exhibit a different instantiation for the language of ElGamal ciphertexts that preserves the original syntax. This approach is an independent contribution that might find broader applicability in neighbouring fields (e.g., for constructions and frameworks that focus on the design of transparent RFs).

Malleable trapdoor smooth projective hash function. In Section 3, we introduce a new primitive that achieves the best of both worlds, i.e, an SPHF that is both malleable and trapdoor. In particular, we focus on designing a M-TSPHF for the language of ElGamal ciphertexts, which will be essential for the design of our srUC PAKE.

In order to construct our M-TSPHF, we augment the TSPHF for ElGamal described above with rerandomization algorithms. In particular, projection keys are rerandomized to $(\gamma g_1^{\widetilde{\alpha_1}} h_1^{\widetilde{\alpha_2}}, \chi_1 g_2^{\tau \widetilde{\alpha_1}}, \chi_2 g_2^{\tau \widetilde{\alpha_2}})$. Because we have lifted the re-

striction of providing msg to hash update algorithms, the hash value can be updated by multiplying it with $e(\mathsf{ct}_1^{\widetilde{\alpha_1}}(\mathsf{ct}_2/\mathsf{msg})^{\widetilde{\alpha_2}}, g_2)$. In Section 3, we cover this construction in further details. In Appendix D, we propose an alternative (less efficient) construction that supports the original malleability syntax.

Our PAKE protocol. In Section 4, we describe our new PAKE protocol in the srUC framework of [19]. This framework introduces cryptographic reverse firewalls in the UC framework as a tool to provide a more fine-grained security analysis of protocols (more details in Section 4.1). In particular, our PAKE protocol remains secure even if one party is malicious *and* the remaining honest party's device is *subverted* with the goal of exfiltrating information throughout the protocol run in an undetectable manner (e.g., by generating messages using rejection sampling or leveraging biased randomness).

As a starting point, we consider the PAKE protocol due to Benhamouda et al. [9], which lies in the CRS-hybrid model[3] while requiring a CCA-secure PKE scheme and a TSPHF for the language of ciphertexts. As remarked before, this TSPHF allows one to compute the same hash value either (i) by using the hashing key, the ciphertext, and the plaintext that was used to generate that ciphertext; or (ii) the projection key, the ciphertext, and the randomness that was used to generate the ciphertext.

The protocol of [9] roughly unfolds as follows. Each party computes a key-pair for the TSPHF, as well as a CCA-secure ciphertext of the password they hold, and forwards the projection key along with the ciphertext to their peer. Upon receiving a tuple (supposedly) from their peer, each party locally computes a session key by using (i) the local hashing key, the ciphertext of their peer, and the local password, and (ii) the projection key received from their peer, the local ciphertext, and the local randomness. The properties of the TSPHF are such that the evaluation of (i) of each party matches with the evaluation of (ii) of its peer, as long as both parties used the same password, and the adversary did not interfere with the protocol run by performing a man-in-the-middle attack.

In order to move to the sanitized setting, we first need to upgrade the primitives used by the original protocol so that RFs can successfully sanitize messages. To this extent, we replace the TSPHF with our newly-introduced M-TSPHF, enabling firewalls to blind outbound projection keys. However, as we see next, sanitizing CCA-secure ciphertexts is less immediate, due to the intrinsic non-malleability of this primitive.

As a first attempt, one could replace the CCA-secure scheme with a replayable chosen-ciphertext attack (RCCA)-secure variant. However, it turns out that this causes the reductions of the original proof of [9] that require CCA security (e.g., in Game 9) to fail. More concretely: in these games, we are attempting

[3] Indeed, it is impossible to UC-realize PAKE without setup [13]. A similar impossibility affects expressive classes of primitives in the context of the UC framework [12,14]. We treat subversion of the CRS-generation procedure as orthogonal to subversion of the protocol parties; F_{CRS} may itself be instantiated by an appropriate subversion-resilient protocol.

to replace the real password w of a honest party with a dummy password $\bar{w}$, so that we obtain a new game that is no longer dependent on the respective party's input. Upon receiving a non-oracle generated ciphertext (i.e., an adversarially-generated message), the reduction of the original proof queries the decryption oracle controlled by the challenger of CCA-security to retrieve the password that was used from the adversary to generate that specific message. In particular, the reduction should make the target honest party output a key computed as per the real protocol whenever the adversary correctly guesses the password. Otherwise, that party should output a random key.

If we replace this challenger with one for RCCA security, the adversary can simply rerandomize the challenge ciphertext to always cause the decryption query to output a special value TEST. The crucial observation is that the same behaviour occurs even if the adversary supplies a non-oracle generated message *without* rerandomizing the challenge ciphertext (and, in particular, by correctly guessing the target party's password). Note that, in the latter case, the reduction should complete the protocol run by outputting a session key that is generated as per the protocol run, whereas in the former case it should output a random session key (as the adversary was indeed unable to guess the password on its own, and it should be unable to determine the key computed from that party). Since it is no longer possible to tell the two settings apart, the reduction will be unable to properly simulate the two hybrids.

We remedy this shortcoming by replacing the CCA-secure scheme with a rerandomizable CPA-secure scheme plus a controlled-malleable NIZK (CM-NIZK). This way, one can first extract the witness and the transformation associated with the ciphertext to determine whether this was rerandomized or not. Then, it can apply CPA security to hybrid away the password from the simulation whenever the message was rerandomized. Note that CPA-security now suffices, as the CM-NIZK allows reductions to determine whether messages are oracle-generated or not (in place of the decryption queries used for the CCA-secure encryption scheme of the original protocol).

With this variation, firewalls can now successfully rerandomize outbound ciphertexts and maul outbound proofs so that they verify under the new statement (i.e., the sanitized ciphertext). Hence, we obtain a PAKE protocol that fully supports sanitization.

We conclude the section by concretely showing the improvements of our PAKE protocol with respect to the subversion-resilient PAKE protocol of [18].

Mauling Groth-Sahai proofs. By the previous paragraph, our srUC PAKE protocol requires CM-NIZKs, which may be instantiated from Groth-Sahai proofs as shown by [20] - the original paper that introduced CM-NIZKs. In particular, the authors of [20] show that Groth-Sahai proofs characterize a class of transformations with respect to which Groth-Sahai proofs are malleable. One can show that the maulings that we are interested in (i.e. ones that arise in the context of CM-NIZK for ElGamal encryption) are supported by [20]'s method. However, their mauling method is too general, and applying it to special cases

may be cumbersome. In more detail, the transformations supported by their method are basic and it is necessary to compose several of them to obtain a transformation of interest (refer to Appendix G to see an example of this limitation). For this reason, we introduce a different method for mauling Groth-Sahai proofs that is directly applicable to our case. In Section 5, we describe our novel mauling method; in Appendix F we use it to instantiate CM-NIZK for ElGamal, in Appendix G, we compare it with existing techniques and demonstrate its applicability beyond our specific use case; and in Appendix H, we present the method in its most general form.

2 Preliminaries

In this section, we summarize the primary cryptographic primitives of interest for this composition, particularly pertaining trapdoor smooth-projective hashing and malleable proof systems.

Notation. $\mathcal{L} = \{L_{\mathsf{lpp},\mathsf{aux}}\}$ represents an NP family of languages parameterized by $(\mathsf{lpp}, \mathsf{aux})$ with witness relation $\mathcal{R} = \{R_{\mathsf{lpp},\mathsf{aux}}\}$. Set is a domain such that, for every $(\mathsf{lpp}, \mathsf{aux})$, $L_{\mathsf{lpp},\mathsf{aux}} \subset \mathsf{Set}$. Usually lpp is sampled by a setup algorithm and aux is sampled adversarially.

2.1 Smooth projective hash function

A smooth projective hash function, or SPHF for short, for $\mathcal{L} = \{L_{\mathsf{lpp},\mathsf{aux}}\}$ is a tuple $(\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda)$ takes the security parameter λ and outputs public parameters pp and a trapdoor τ[4]. We always implicitly assume that included in pp, there is an lpp that together with aux, specifies a language $L_{\mathsf{lpp},\mathsf{aux}}$. We also implicitly assume that pp includes the security parameter.
- $\mathsf{HashKG}(\mathsf{pp})$ takes public parameters pp as input and generates hashing key hk.
- $\mathsf{ProjKG}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk})$ takes public parameters pp, auxiliary input aux and a hashing key hk as input and outputs a projection key hp.[5]
- $\mathsf{Hash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}, x)$, given public parameters pp, auxiliary input aux, a hashing key hk and a problem instance x, outputs a hash value hv.

[4] To ensure notational consistency with the syntax of [9], the output of Setup may contain a trapdoor τ related to the public parameters pp. Concretely, Setup may implicitly run a CRS generation algorithm.

[5] There is a variant of this notion that lets the projection key depend on the statement x. This variant is called GL-type [26]. Here, we consider the so-called KV-type instead [31], since in our PAKE protocol each party generates its hashing and projection keys before receiving the peer's ciphertext x; hence the projection key cannot depend on that statement

– $\mathsf{ProjHash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hp}, x, w)$ takes public parameters pp, auxiliary input aux, projection key hp, a problem instance x and a witness w as input and outputs a hash value hv.

Informally, a SPHF is correct if the evaluation of Hash and $\mathsf{ProjHash}$ are equivalent for instances in the language. A SPHF is (computationally) smooth if no efficient adversary holding hp can distinguish the output of Hash from uniform for instances not in the language with non-negligible probability.

In what follows, we show definitions for correctness and different flavours of smoothness. We implicitly assume pp is implicitly provided to all of the SPHF algorithms:

– **Correctness:** For every aux, and for every $(x, w) \in R_{\mathsf{lpp},\mathsf{aux}}$:

$$\mathbb{P}\left[\begin{array}{c|l} \mathsf{Hash}(\mathsf{aux}, \mathsf{hk}, x) & (\mathsf{pp}, \tau) \leftarrow_{\$} \mathsf{Setup}(1^\lambda) \\ = & \mathsf{hk} \leftarrow_{\$} \mathsf{HashKG}() \\ \mathsf{ProjHash}(\mathsf{aux}, \mathsf{hp}, x, w) & \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{aux}, \mathsf{hk}) \end{array}\right] = 1$$

– **Perfect smoothness:** For every aux and for every $x \notin R_{\mathsf{lpp},\mathsf{aux}}$, the following distributions are perfectly indistinguishable, where $\mathcal{H}$ is the space of hash values:

$$\left[\begin{array}{c|l} (\mathsf{hv}, \mathsf{hp}) & (\mathsf{pp}, \tau) \leftarrow_{\$} \mathsf{Setup}(1^\lambda) \\ & \mathsf{hk} \leftarrow_{\$} \mathsf{HashKG}() \\ & \mathsf{hv} \leftarrow_{\$} \mathcal{H} \\ & \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{aux}, \mathsf{hk}) \end{array}\right] \equiv \left[\begin{array}{c|l} (\mathsf{hv}, \mathsf{hp}) & (\mathsf{pp}, \tau) \leftarrow_{\$} \mathsf{Setup}(1^\lambda) \\ & \mathsf{hk} \leftarrow_{\$} \mathsf{HashKG}() \\ & \mathsf{hv} \leftarrow \mathsf{Hash}(\mathsf{aux}, \mathsf{hk}, x) \\ & \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{aux}, \mathsf{hk}) \end{array}\right]$$

– **Computational smoothness:** For every PPT adversary $\mathcal{A}$ and for every aux:

$$\mathbb{P}\left[\begin{array}{c|l|l} b = b' & & b \leftarrow_{\$} \{0,1\} \\ & (\mathsf{pp}, \tau) \leftarrow_{\$} \mathsf{Setup}(1^\lambda) & \text{if } x \in L_{\mathsf{lpp},\mathsf{aux}} \text{ or } b = 0 \\ & x \leftarrow \mathcal{A}(\tau, \mathsf{pp}) & \quad H \leftarrow \mathsf{Hash}(\mathsf{aux}, \mathsf{hk}, x) \\ & \mathsf{hk} \leftarrow_{\$} \mathsf{HashKG}() & \text{else} \\ & \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{aux}, \mathsf{hk}) & \quad H \leftarrow_{\$} \mathcal{H} \\ & & b' \leftarrow \mathcal{A}(\mathsf{hp}, H) \end{array}\right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda)$$

2.2 Trapdoor smooth projective hash function

A trapdoor SPHF, or TSPHF for short, for $\mathcal{L} = \{L_{\mathsf{lpp},\mathsf{aux}}\}$ consists of a tuple of algorithms $(\mathsf{Setup}, \mathsf{TSetup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}, \mathsf{VerHP}, \mathsf{THash})$. Among these algorithms, Setup and HashKG have the same syntax as in the SPHF definition. The only syntactical difference that the algorithms ProjKG, Hash and $\mathsf{ProjHash}$ have with their counterparts in the definition of SPHF is that they take an extra input crs' that is generated by TSetup. The rest of the algorithms have the following syntax.

– $\mathsf{TSetup}(\mathsf{pp})$, given the public parameters pp, outputs a common reference string crs' and a trapdoor τ'.

– $\mathsf{VerHP}(\mathsf{pp}, \mathsf{aux}, \mathsf{crs}', \mathsf{hp})$ takes public parameters pp, auxiliary input aux, common reference string crs' and a projection key hp and outputs 1 to indicate that hp is valid and outputs 0 to indicate the opposite.
– $\mathsf{THash}(\mathsf{pp}, \mathsf{aux}, \mathsf{crs}', \tau', \mathsf{hp}, x)$ takes public parameters pp, auxiliary input aux, common reference string crs', trapdoor τ', projection key hp and a problem instance x and outputs a hash value hv.

Informally, TSetup generates additional public parameters crs' that enable the verification of projection keys against the public parameters by means of VerHP. Moreover, TSetup outputs an *additional* trapdoor τ' (which is distinct from the trapdoor τ output by Setup) that can be leveraged to evaluate the hash of an element x in a consistent manner to the evaluation of Hash by only using a projection key hp (i.e., without having access to its related witness).

In the remainder of the paper, the TSPHF algorithms are assumed to implicitly receive pp and crs' as inputs. We summarize the properties of this primitive in what follows:

– **Correctness**
  • **Hash correctness** is similar to SPHFs, with the difference that TSetup is called in the setup phase and crs' is given to the algorithms that require it.
  • **Trapdoor correctness:** For every aux and every $x \in \mathsf{Set}$:

$$\mathbb{P}\left[\begin{array}{c|l} \mathsf{VerHP}(\mathsf{aux}, \mathsf{hp}) = 1 \\ \wedge \\ H_1 = H_2 & \begin{array}{l} (\mathsf{pp}, \tau) \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{hk} \leftarrow \mathsf{HashKG}() \\ (\mathsf{crs}', \tau') \leftarrow \mathsf{TSetup}() \\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{aux}, \mathsf{hk}) \\ H_1 \leftarrow \mathsf{Hash}(\mathsf{aux}, \mathsf{hk}, x) \\ H_2 \leftarrow \mathsf{THash}(\mathsf{aux}, \tau', \mathsf{hp}, x) \end{array} \end{array}\right] = 1$$

– **Smoothness** is the same as SPHFs, with the difference that TSetup is called in the setup phase and the output crs' is given to the algorithms that require it and also to the adversary (in both computational and perfect settings).
– **Soundness:** For every PPT adversary $\mathcal{A}$:

$$\mathbb{P}\left[\begin{array}{c|l} (x, w) \in R_{\mathsf{lpp},\mathsf{aux}} \\ \wedge\ \mathsf{VerHP}(\mathsf{aux}, \mathsf{hp}) = 1 \\ \wedge\ \mathsf{THash}(\mathsf{aux}, \tau', \mathsf{hp}, x) \\ \neq \mathsf{ProjHash}(\mathsf{aux}, \mathsf{hp}, x, w) & \begin{array}{l} (\mathsf{pp}, \tau) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{crs}', \tau') \leftarrow \mathsf{TSetup}() \\ (\mathsf{hp}, \mathsf{aux}, x, w) \leftarrow \mathcal{A}(\mathsf{pp}, \tau, \mathsf{crs}') \end{array} \end{array}\right] \leqslant \mathsf{negl}(\lambda)$$

2.3 Proof systems

A proof system for an NP language L with the corresponding NP relation R is a tuple $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ of algorithms in which $\mathsf{CRSSetup}(1^\lambda)$ takes the security parameter and outputs a common reference string σ, $\mathcal{P}(\sigma, x, w)$ takes the common reference string σ, statement x and witness w and outputs a proof π, and $\mathcal{V}(\sigma, x, \pi)$ takes the common reference string, statement x and proof π and

outputs 1 or 0 to indicate that the proof is accepted or not. We refer to Appendix A for formal definitions of traditional properties we consider within this composition (in particular: completeness, soundness, zero-knowledge, witness-indistinguishability, proof-of-knowledge, re-randomizability). In what follows, we report more peculiar properties pertaining malleability.

Malleable and strong derivation-private proof systems. Informally, a *malleable* proof system allows transforming an accepting proof π for a statement x to an accepting proof π' for a (potentially different) statement x', where x' is obtained from x by applying some transformation $T = (T_x, T_w)$. This transformation should be *admissible*, meaning that any $(x, w) \in R$ should be mapped to a $(x', w') \in R$, with $(x', w') = (T_x(x), T_w(w))$. We denote sets of allowable transformations with $\mathcal{T}$. A *strong-derivation private* proof system ensures that mauled proofs should be indistinguishable from freshly-simulated proofs. These properties are captured as follows:

- **Malleability:** Let $\mathcal{T}$ be a set of transformations over L. A proof system is said to be *malleable* with respect to $\mathcal{T}$ if there exists an algorithm ZKEval such that, for all crs, x, π such that $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$, the following holds:

$$\mathbb{P}\left[T \in \mathcal{T} \wedge \ \mathcal{V}(\mathsf{crs}, T(x), \widetilde{\pi}) = 1 \ \middle|\ \widetilde{\pi} \leftarrow \mathsf{ZKEval}(\mathsf{crs}, T, \pi) \right] = 1$$

- **Strong derivation privacy:** A malleable proof system is said to be *strongly derivation private* if there exists a PPT algorithm $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ (i.e., a simulator) such that, for every PPT adversary $\mathcal{A}$, the following holds:

$$\mathbb{P}\left[\begin{array}{c} T \in \mathcal{T} \\ \wedge\ \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \\ \wedge\ b = b' \end{array} \middle| \begin{array}{l} (\mathsf{crs}, \tau_s) \leftarrow \mathcal{S}_0(1^\lambda) \\ (x, w, \pi, T) \leftarrow \mathcal{A}(\mathsf{crs}, \tau_s) \\ b \leftarrow\!\!\$\ \{0,1\} \\ \widetilde{\pi} \leftarrow\!\!\$ \begin{cases} \mathsf{ZKEval}(\mathsf{crs}, T, \{x, \pi\}) & \text{if } b = 0 \\ \mathcal{S}_1(\mathsf{crs}, T_x(x), \tau_s) & \text{if } b = 1 \end{cases} \\ b' \leftarrow \mathcal{A}(\widetilde{\pi}) \end{array} \right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda)$$

Controlled-malleable simulation-sound extractability. In the context of malleable proof systems, certain desirable standard security properties are no longer meaningful. One such example is *simulation-sound extractability* (SSE), which informally states that a knowledge extractor should still work even if the adversary can query the simulator on arbitrary statements to obtain simulated proofs. By introducing malleability in a proof system for an NP-language $\mathcal{L}$, the adversary could simply query the simulator to obtain an accepting proof π on $x \notin \mathcal{L}$, and then maul π with some allowable transformation to obtain an accepting proof π' for which the extractor will fail to extract (as no witness exists for such a statement).

For that, [20] coined the weaker notion of *controlled-malleable* SSE (CM-SSE), which essentially adapts SSE to account for the malleability of proof

systems. In particular, it is acceptable for the extractor to either output a witness, or a previously-queried statement alongside an allowable transformation $T = (T_x, T_w) \in \mathcal{T}$. This way, if the adversary attempts the aforementioned attack, the extractor will successfully extract a tuple (x, T), where x is a previously-queried statement such that $x' = T_x(x)$ (i.e., the adversary queried the simulator on x and mauled the respective proof and statement with allowable transformation $T \in \mathcal{T}$). We provide a formal definition in what follows.

Definition 1 (Controlled-malleable simulation-sound extractability [20]). *Let Π be a malleable proof system that additionally satisfies soundness, zero-knowledge, and proof-of-knowledge, and let $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ and $\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1)$ be, respectively, the simulator and extractor algorithms promised by zero-knowledge and proof-of-knowledge. Let $\mathcal{SE} = (\mathcal{SE}_0, \mathcal{SE}_1)$ be an algorithm that, on input 1^λ, returns $(\mathsf{crs}, \tau, \xi)$ such that (crs, τ) is identically distributed to the output of $\mathcal{S}_0$, where ξ denotes the extraction trapdoor. We say Π enjoys controlled-malleable simulation-sound extractability if, for every PPT adversary $\mathcal{A}$, the following holds:*

$$\Pr\left[\begin{array}{c|l} \begin{array}{c}\mathcal{V}(\mathsf{crs}, x', \pi') = 1 \wedge (x', \pi') \notin Q \\ \wedge \\ (w' \neq \bot \wedge (x', w') \notin R_{\mathcal{L}} \\ \vee \\ (x, T) \neq (\bot, \bot) \wedge \\ (x \notin Q_{\mathsf{stmt}} \vee x' \neq T_x(x) \vee T \notin \mathcal{T}) \\ \vee \\ (w', x, T) = (\bot, \bot, \bot))\end{array} & \begin{array}{l}(\mathsf{crs}, \tau, \xi) \leftarrow \mathcal{SE}_0(1^\lambda) \\ (x', \pi') \leftarrow \mathcal{A}^{\mathcal{S}_1(\mathsf{crs}, \tau, \cdot)}(\mathsf{crs}, \xi) \\ (w', x, T) \leftarrow \mathcal{E}_1(\mathsf{crs}, \xi, x, \pi)\end{array}\end{array}\right] \leqslant \mathsf{negl}(\lambda),$$

where Q contains the replies of the oracle $\mathcal{S}_1$, and Q_{stmt} contains the queried statements. In the remainder of the paper, whenever we refer to a CM-NIZK, we mean a CM-SSE NIZK with straight-line extraction using the extraction trapdoor.

3 Malleable trapdoor smooth projective hash functions

In this section, we define a new variant of malleable SPHF, formalize the notion of *malleable trapdoor SPHF* (M-TSPHF) and instantiate a specific M-TSPHF for the language of ElGamal ciphertexts. Looking ahead, this will be a core building block of our subversion-resilient PAKE protocol and its related instantiation.

Section 3.1 introduces our variant of malleable SPHF and highlights its differences with the original definition. In Section 3.2, we formally define our new primitive, and in Section 3.3, we construct an M-TSPHFs for the language of ElGamal ciphertexts.

3.1 Malleable smooth projective hash function

Malleable SPHFs (MSPHFs), were first introduced in [21]. Roughly speaking, a MSPHF adds mauling (or rerandomization) capabilities to a normal SPHF,

so that one may maul (or rerandomize) projection keys (or language elements) in a way that (computationally) hides the original projection key (or language element), while making it possible to update the hash value corresponding to the original key (or element) to a hash value for the new key (or element), using the randomness used in the mauling (or rerandomization) process.

We report the original syntax of [21] in Appendix A.1. Notably, the hash update algorithms (i.e., MaulH and ReranH) do not take aux as input. This limitation stems from the specific use-case that was considered in [21], where MSPHFs are used to deploy *transparent* cryptographic reverse firewalls that sanitize outbound keys (or elements) and run the respective hash update algorithm to ensure correctness of protocols.

However, for certain applications, this restriction can be unnecessary. Concretely, in the srUC framework, a firewall is allowed to return its randomness to the core it is protecting; hence, it makes little sense to forbid the core (who is in possession of aux) from running MaulH (or ReranH) to adjust its hash value locally without leveraging this additional information. Looking ahead, lifting this restriction from the syntax will allow us to design more efficient M-TSPHFs for suitable languages. Concretely, the M-TSPHF for ElGamal we outlined in our technical overview is significantly more performing compared to the alternative construction that adheres to the original syntax we present in Appendix D. These performance improvements are, in turn, reflected in the construction of our PAKE protocol. In what follows, we formally define our revised syntax that allows MaulH and ReranH to take aux as inputs.

A MSPHF is a tuple of algorithms $(\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}, \mathsf{MaulK}, \mathsf{MaulHK}, \mathsf{MaulH}, \mathsf{ReranE}, \mathsf{ReranH}, \mathsf{SampR}, \mathsf{SampW})$ such that $(\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$ form an SPHF and the new seven algorithms have the following syntax:

- $\mathsf{MaulK}(\mathsf{pp}, \mathsf{hp}, \widetilde{r})$ takes the public parameters, a projection key hp and a randomness $\widetilde{r}$ and outputs a new projection key $\widetilde{\mathsf{hp}}$.
- $\mathsf{MaulH}(\mathsf{pp}, \mathsf{aux}, \mathsf{hp}, x, \widetilde{r})$ takes as input the public parameters, auxiliary input aux, a projection key hp, a word x, and a randomness $\widetilde{r}$ and outputs a hash value $\widetilde{\mathsf{hv}}$.
- $\mathsf{MaulHK}(\mathsf{pp}, \mathsf{hk}, \widetilde{r})$ takes the public parameters, a hashing key hk and a randomness $\widetilde{r}$ and outputs a new hashing key $\widetilde{\mathsf{hk}}$.
- $\mathsf{ReranE}(\mathsf{pp}, x, \widetilde{w})$ takes as input the public parameters pp, a word x and a randomness $\widetilde{w}$ and outputs a new word $\widetilde{x}$.
- $\mathsf{ReranH}(\mathsf{pp}, \mathsf{aux}, \mathsf{hp}, x, \widetilde{w})$ takes the public parameters pp, auxiliary input aux, a projection key hp, a word x and a randomness $\widetilde{w}$ and outputs a hash value $\widetilde{\mathsf{hv}}$.
- $\mathsf{SampR}(\mathsf{pp})$ takes the public parameters pp and outputs a randomness $\widetilde{r}$.
- $\mathsf{SampW}(\mathsf{pp})$ takes the public parameters pp and outputs a randomness $\widetilde{w}$.

The security properties of our alternative syntax are defined exactly as in the original definition, which is reported in Appendix A.1. In the remainder of the paper, the MSPHF algorithms implicitly receive pp as input.

3.2 Malleable TSPHFs

A malleable TSPHF, or M-TSPHF for short, for $\{L_{\mathsf{lpp},\mathsf{aux}}\}$ consists of a tuple of algorithms $(\mathsf{Setup}, \mathsf{TSetup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}, \mathsf{THash}, \mathsf{VerHP}, \mathsf{MaulK}, \mathsf{MaulHK}, \mathsf{MaulH}, \mathsf{ReranE}, \mathsf{ReranH}, \mathsf{SampW}, \mathsf{SampR})$ such that $(\mathsf{Setup}, \mathsf{TSetup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}, \mathsf{THash}, \mathsf{VerHP})$ is a TSPHF and the seven algorithms $(\mathsf{MaulK}, \mathsf{MaulHK}, \mathsf{MaulH}, \mathsf{ReranE}, \mathsf{ReranH}, \mathsf{SampR}, \mathsf{SampW})$ have the same syntax as in malleable SPHFs, except that they also accept as input crs', output by TSetup. Security properties about the mauling part of the malleable TSPHF are similar to the corresponding properties of MSPHFs, with the difference that TSetup is always run and the output crs' is passed to the adversary, and the projection keys output by the adversary are verified by VerHP. For concreteness, we summarize the properties in what follows. Note that pp and crs' are implicitly provided to all algorithms, and it is assumed that $*$ is an operation over the space of hash values.

– **Key Indistinguishability**. For any PPT adversary $\mathcal{A}$:

$$\Pr\left[\begin{array}{c|l} b = b' & (\mathsf{pp},\tau) \leftarrow_{\$} \mathsf{Setup}(1^\lambda) \\ \wedge & (\mathsf{crs}',\tau') \leftarrow_{\$} \mathsf{TSetup}() \\ \mathsf{VerHP}(\mathsf{aux},\mathsf{hp}_1) & (\mathsf{hp}_1,\mathsf{hp}_2) \leftarrow \mathcal{A}(\mathsf{pp},\mathsf{crs}') \\ \wedge & b \leftarrow_{\$} \{0,1\} \\ \mathsf{VerHP}(\mathsf{aux},\mathsf{hp}_2) & \widetilde{r} \leftarrow_{\$} \mathsf{SampR}() \\ & \widetilde{\mathsf{hp}} \leftarrow \mathsf{MaulK}(\mathsf{hp}_b,\widetilde{r}) \\ & b' \leftarrow \mathcal{A}(\widetilde{\mathsf{hp}}) \end{array}\right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda)$$

– **Projection Consistency**. For every $x \in \mathsf{Set}$ and every aux:

$$\Pr\left[\begin{array}{c} \mathsf{Hash}(\mathsf{aux},\widetilde{\mathsf{hk}},x) \\ \neq \mathsf{Hash}(\mathsf{aux},\mathsf{hk},x) * \widetilde{\mathsf{hv}} \\ \text{or } \mathsf{VerHP}(\mathsf{aux},\widetilde{\mathsf{hp}}) = 0 \end{array}\middle|\begin{array}{l} (\mathsf{pp},\tau) \leftarrow_{\$} \mathsf{Setup}(1^\lambda) \\ \mathsf{hk} \leftarrow_{\$} \mathsf{HashKG}() \\ (\mathsf{crs}',\tau') \leftarrow_{\$} \mathsf{TSetup}() \\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{aux},\mathsf{hk}) \\ \widetilde{r} \leftarrow_{\$} \mathsf{SampR}() \\ \widetilde{\mathsf{hp}} \leftarrow \mathsf{MaulK}(\mathsf{hp},\widetilde{r}) \\ \widetilde{\mathsf{hv}} \leftarrow \mathsf{MaulH}(\mathsf{aux},\mathsf{hp},x,\widetilde{r}) \\ \widetilde{\mathsf{hk}} \leftarrow \mathsf{MaulHK}(\mathsf{hk},\widetilde{r}) \end{array}\right] \leqslant \mathsf{negl}(\lambda)$$

– **Element Indistinguishability**. For any PPT adversary $\mathcal{A}$:

$$\Pr\left[b = b' \middle| \begin{array}{ll} \mathsf{pp} \leftarrow_{\$} \mathsf{Setup}(1^\lambda) & b \leftarrow_{\$} \{0,1\} \\ (\mathsf{crs}',\tau') \leftarrow_{\$} \mathsf{TSetup}() & \widetilde{w} \leftarrow_{\$} \mathsf{SampW}() \\ (x_1,x_2) \leftarrow \mathcal{A}(\mathsf{pp},\mathsf{crs}') & \widetilde{x} \leftarrow \mathsf{ReranE}(x_b,\widetilde{w}) \\ & b' \leftarrow \mathcal{A}(\widetilde{x}) \end{array}\right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda)$$

– **Rerandomization Consistency**. For every $x \in \mathsf{Set}$ and aux:

$$\Pr\left[\begin{array}{c} \mathsf{Hash}(\mathsf{aux},\mathsf{hk},\widetilde{x}) \\ \neq \mathsf{Hash}(\mathsf{aux},\mathsf{hk},x) * \widetilde{\mathsf{hv}} \end{array} \middle| \begin{array}{l} (\mathsf{pp},\tau) \leftarrow_{\$} \mathsf{Setup}(1^\lambda) \\ (\mathsf{crs}',\tau') \leftarrow_{\$} \mathsf{TSetup}() \\ \mathsf{hk} \leftarrow_{\$} \mathsf{HashKG}() \\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{aux},\mathsf{hk}) \\ \widetilde{w} \leftarrow_{\$} \mathsf{SampW}() \\ \widetilde{x} \leftarrow \mathsf{ReranE}(x,\widetilde{w}) \\ \widetilde{\mathsf{hv}} \leftarrow \mathsf{ReranH}(\mathsf{aux},\mathsf{hp},x,\widetilde{w}) \end{array}\right] \leqslant \mathsf{negl}(\lambda)$$

– **Membership Preservation** is similar to the malleable SPHF definition.

3.3 M-TSPHF for ElGamal

Next, we construct a M-TSPHF for the language of ElGamal ciphertexts. The TSPHF part of this construction is based on [9]'s generic framework for TSPHFs, which we extend to include mauling and rerandomization algorithms.

First, let us clearly specify the language of ElGamal ciphertexts. Assume that group $\mathbb{G}_0$ of prime order p with generator g_0 is given. For $\mathsf{aux} = m \in \mathbb{G}_0$ and $\mathsf{lpp} = h = g_0^s$, where $s \in \mathbb{Z}_p$, let $L_{\mathsf{lpp},\mathsf{aux}} = \{(c_1,c_2) \mid \exists w \in \mathbb{Z}_p \ s.t. \ (c_1,c_2) = (g_0^w, mh^w)\}$. The language of ElGamal ciphertexts is specified by $\mathcal{L} = \{L_{\mathsf{lpp},\mathsf{aux}}\}$.

Construction 1 *Let $\mathbb{G}_0$, $\mathbb{G}_1$ and $\mathbb{G}_T$ be groups of order p, $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a bilinear pairing and assume that g_0 and g_1 are generators for $\mathbb{G}_0$ and $\mathbb{G}_1$ respectively.*

– ***TSPHF part:***
- $\mathsf{Setup}(1^\lambda)$*: Samples* $s \leftarrow \mathbb{Z}_p$ *and outputs* $\mathsf{pp} = (\lambda, h_0 = g_0^s)$.
- $\mathsf{HashKG}(\mathsf{pp})$*: Outputs* $\mathsf{hk} = (\alpha_1,\alpha_2) \leftarrow \mathbb{Z}_p^2$.
- $\mathsf{TSetup}(\mathsf{pp})$*: Samples* $\tau' \leftarrow \mathbb{Z}_p$ *and outputs* $(\tau', \zeta = g_1^{\tau'})$.
- $\mathsf{ProjKG}(\mathsf{pp},\mathsf{aux},\zeta,\mathsf{hk})$*: sets* $\gamma = g_0^{\alpha_1} h_0^{\alpha_2}$, $\chi = (\chi_1 = g_1^{\tau'\alpha_1}, \chi_2 = g_1^{\tau'\alpha_2})$ *and outputs* $\mathsf{hp} = (\gamma,\chi)$.
- $\mathsf{Hash}(\mathsf{pp},\mathsf{aux},\zeta,\mathsf{hk},C = (\mathsf{ct}_1,\mathsf{ct}_2)) = e(\mathsf{ct}_1^{\alpha_1}(\mathsf{ct}_2/\mathsf{aux})^{\alpha_2}, g_1)$.
- $\mathsf{ProjHash}(\mathsf{pp},\mathsf{aux},\zeta,\mathsf{hp},C,w)$*: outputs* $e(\gamma^w, g_1)$.
- $\mathsf{THash}(\mathsf{pp},\mathsf{aux},\zeta,\tau',\mathsf{hp},C)$*: outputs* $(e(g_0,\chi_1)e(h_0,\chi_2))^{\tau'^{-1}}$.
- $\mathsf{VerHP}(\mathsf{pp},\mathsf{aux},\zeta,\mathsf{hp})$*: Verifies that* $\chi \in \mathbb{G}_1^2$ *and* $e(\gamma,\zeta) = e(g_0,\chi_1)e(h_0,\chi_2)$.

– ***Malleability part.***
- $\mathsf{SampR}(\mathsf{pp})$*: Outputs* $\widetilde{r} = (\widetilde{\alpha_1},\widetilde{\alpha_2}) \leftarrow \mathbb{Z}_p^2$.
- $\mathsf{MaulK}(\mathsf{pp},\zeta,\mathsf{hp},\tilde{r})$*: Outputs* $(\gamma g_0^{\widetilde{\alpha_1}} h_0^{\widetilde{\alpha_2}}, \chi_1 g_1^{\tau'\widetilde{\alpha_1}}, \chi_2 g_1^{\tau'\widetilde{\alpha_2}})$.
- $\mathsf{MaulH}(\mathsf{pp},\mathsf{aux},\zeta,\mathsf{hp},C,\tilde{r})$*: Outputs* $e(\mathsf{ct}_1^{\widetilde{\alpha_1}}, g_1)e((\mathsf{ct}_2/\mathsf{aux})^{\widetilde{\alpha_2}}, g_1)$.
- $\mathsf{SampW}(\mathsf{pp})$*: Outputs* $\widetilde{w} \leftarrow \mathbb{Z}_p$.
- $\mathsf{ReranE}(\mathsf{pp},\zeta,C,\widetilde{w})$*: Outputs* $(\mathsf{ct}_1 g_0^{\widetilde{w}}, \mathsf{ct}_2 h_0^{\widetilde{w}})$.
- $\mathsf{ReranH}(\mathsf{pp},\mathsf{aux},\zeta,\mathsf{hp},C,\widetilde{w})$*: Outputs* $e(\gamma^{\widetilde{w}}, g_1)$.

Theorem 1. *Construction 1 is a M-TSPHF for the language of ElGamal ciphertexts.*

See proof in Appendix B

4 Subversion-resilient UC PAKE

Now, we give a concise overview of the subversion-resilient UC framework introduced by Chakraborty et al. [19], formally present our subversion-resilient PAKE protocol, and provide a comparison with respect to the subversion-resilient PAKE protocol of Chakraborty et al. [18]. Our construction is derived by sanitizing the PAKE-from-TSPHF approach of Benhamouda et al. [9].

4.1 A brief introduction to subversion-resilient UC

At its core, the framework of subversion-resilient universal composability (srUC) of Chakraborty et al. [19] extends the framework of universal composability (UC) of Canetti [10] to support cryptographic reverse firewalls. In what follows, we provide a bird's eye view on the main novelties that have been introduced.

Corruptions. The main purpose of reverse firewalls consists in thwarting so-called *subversion attacks*. [19] focuses on a specific class of subversions denoted as *specious corruptions*, which is characterized by the notion of *black-box testing*. Let C be an entity implementing some protocol Π, and let $\tilde{\mathsf{C}}$ be an entity implementing a subverted implementation of Π that is sampled from some efficiently sampleable distribution D. Let T be a PPT algorithm that has non-rewinding black-box access to either of the entities. We say that $\tilde{\mathsf{C}}$ is specious if no PPT test T can determine whether it is interacting with C or $\tilde{\mathsf{C}}$. This is formalized by the following game:

- Sample $(\tilde{\mathsf{C}}, a) \leftarrow \mathsf{D}$, where a is an auxiliary string shared with the adversary.
- Sample $b \leftarrow_{\$} \{0,1\}$.
  - If $b = 0$, run T with black-box access to $\tilde{\mathsf{C}}$.
  - If $b = 1$, run T with black-box access to C.
- Let b' be the output of T. Output $b = b'$.

Definition 2 (Specious subversion [19]). *Let* $\mathtt{Test}_{\mathsf{D},\mathsf{T}}$ *be the probability that the guess of the game above is correct.* D *is computationally specious if, for all PPT tests* T*, it holds that* $|\mathtt{Test}_{\mathsf{D},\mathsf{T}} - 1/2| < \mathsf{negl}(\lambda)$.

As a concrete example, [19, Lemma 1] considers a C implementing a protocol Π that outputs fresh uniform random strings. A specious core could be one whose output is generated by evaluating a pseudo-random permutation with key k on a random value that gets incremented for every evaluation, with k being revealed to the adversary upon deployment of the specious core. The adversary can now determine whether the implementation is subverted or not by querying $\tilde{\mathsf{C}}$ twice.

As stressed in [19], introducing a specious distribution D that is quantified *after* the test is necessary to avoid trivializing the notion of specious corruption, which is crucially defined over all possible tests. In the example above, if we restrict our attention to a single specious core, there could always exist a test

that guesses the trapdoor key k and runs the adversarial strategy to determine whether the core is subverted.

One important detail that is implicit in all the protocols and results of [19] is that all specious implementations should additionally be "functionality-maintaining", meaning that their behaviour should be indistinguishable to that of an honest implementation for *all* inputs and input messages. This follows the spirit of "signalling subversions" considered in [19, Definition 2], which consists in covertly exfiltrating information throughout a protocol run. Indeed, the example we reported above is specious *and* functionality-maintaining.

As a commonality with other definitions in the literature (e.g., [16]), this characterization of subversion attacks rules out a specific class of input-trigger attacks that may cause the specious core to behave arbitrarily upon receiving a specific triggering message (e.g., by crashing). The only exceptions appear to be [18,33], as all the protocols thereby defined additionally sanitize all inbound messages. While this approach appears to be sufficient to comprehensively handle triggering inputs, this property is not formalized yet in the srUC framework [19].

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

Table 1: The corruption translation table of [19]. The highlighted rows represent the sufficient cases to be considered within security proofs.

Modelling parties. The first important difference compared to traditional UC comes in the modelling of parties. Each party P is composed of a core C and a reverse firewall F. The former runs the protocol; the latter accesses and sanitizes the traffic of the core it is protecting. Crucially, firewalls aren't allowed to have any knowledge of their respective core's inputs. These two components can be corrupted independently: cores can be Honest, Specious, or Malicious; firewalls can be Honest, SemiHonest, or Malicious. The joint behaviour of the two components can be mapped to that of a regular party. For example, if a core is honest and its firewall is semi-honest, the overall party should still be honest, as the firewall should not degrade the security of its honest core. We report all possible combinations of this mapping in Table 1. As shown in [19, Section 2.3], certain corruption combinations are redundant within security proofs.

In order to further simplify security proofs, [19] defines the notion of *strongly sanitizing* firewalls, which captures the fact that a protocol run with specious cores protected by honest firewalls is indistinguishable from a protocol run with

honest cores protected by honest firewalls. This way, as long as firewalls are doing their job properly, one does not have to explicitly consider the case of SPECIOUS core and HONEST firewall within security proofs.

Finally, [19] remarks that it may still be possible to prove some partial security for the ISOLATE case (e.g., that the inputs of the honest cores are not leaked throughout the protocol run). However, the joint behaviour of this party is still tantamount to a malicious party, given that the firewall can behave arbitrarily. Hence, this case can also be safely dropped from the analysis, leaving only (i) HONEST core and SEMIHONEST firewall, and (ii) MALICIOUS core and MALICIOUS firewall.

Ideal functionalities. Splitting parties into two components comes with additional modelling choices for ideal functionalities, which are separated in *sanitizable* and *regular* ideal functionalities.

Sanitizable ideal functionalities explicitly interact with cores and firewalls, exposing, for every party P, an input-output interface IO interacting with C, and a sanitation interface S interacting with F. As a concrete example, the sanitizable string commitment ideal functionality $\mathcal{F}_{\mathsf{sCOM}}$ [19, Section 3] allows the committer's firewall to blind the input of the committer's core.

On the other hand, regular ideal functionalities match traditional UC ideal functionalities, exposing an interface for each party. Note that it is still perfectly reasonable to design subversion-resilient protocols that realize regular ideal functionalities. For instance, [19, Section 4] shows an instantiation of the regular coin tossing ideal functionality $\mathcal{F}_{\mathsf{TOSS}}$, and in this work we instantiate the regular $\mathcal{F}_{\mathsf{PAKE}}$ functionality.

As a technical detail, whenever one proves security for a protocol realizing a regular ideal functionality, [19] defines a variant of srUC security dubbed *wrapped* subversion-resilient UC (wsrUC) security. The only difference introduced by the latter consists in adding a "wrapper" to $\mathcal{F}$ that translates corruptions following Table 1 and forwards messages between the parties' components and $\mathcal{F}$. This is necessary, as otherwise the environment would always have a trivial distinguishing advantage: the real-world protocol has an explicit separation of cores and firewalls, whereas $\mathcal{F}$ only interacts parties.

Communication channels. From a formal standpoint, a core and a firewall constituting a party are separate entities that cannot communicate with each other. For that, [19] assumes as a building block the ideal functionality for sanitizable authenticated communication $\mathcal{F}_{\mathsf{SAT}}$, which provides (1) secure and authenticated channels between cores and their firewall, and (2) authenticated channels between firewalls. More recently, [18] extended the srUC framework to the unauthenticated setting, replacing $\mathcal{F}_{\mathsf{SAT}}$ with a backbone of communication that only requires item (1). This is the minimal security assumption in the context of reverse firewalls, as otherwise the adversary would gain access to the channel between cores and their firewall, nullifying any potential sanitation.

4.2 The PAKE ideal functionality

In Figure 1, we provide a description of $\mathcal{F}_{\mathsf{PAKE}}$ that adapts the description of [18].

$\mathcal{F}_{\mathsf{PAKE}}$ interacts with a set of parties $\mathcal{P}$ and an adversary $\mathcal{S}$ via the following queries:

Upon receiving a query $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', \mathsf{pw})$ **from** P**:**
Record $(\mathsf{P}, \mathsf{P}', \mathsf{pw})$, mark it as fresh, and leak $(\mathsf{sid}, \mathsf{P}, \mathsf{P}')$ to $\mathcal{S}$. Ignore all future messages from P.

Upon receiving a query $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, \tilde{\mathsf{pw}})$ **from the adversary** $\mathcal{S}$**:**
If there exists a record of the form $(\mathsf{P}, \cdot, \mathsf{pw})$ which is fresh, then:
- If $\tilde{\mathsf{pw}} = \mathsf{pw}$, mark the record as $\mathsf{compromised}$ and return "correct guess" to $\mathcal{S}$.
- If $\tilde{\mathsf{pw}} \neq \mathsf{pw}$, mark the record as $\mathsf{interrupted}$ and return "wrong guess" to $\mathcal{S}$.

Upon receiving a query $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, K)$ **from** $\mathcal{S}$**, where** $|K| = \lambda$**:**
If there is a record of the form $(\mathsf{P}, \mathsf{P}', \mathsf{pw})$ that is not marked as $\mathsf{completed}$, then:
- If any of the following conditions hold, output (sid, K) to party P:
  - P is corrupted.
  - This record is fresh, there exists a record $(\mathsf{P}', \mathsf{P}, \mathsf{pw}')$ with $\mathsf{pw}' = \mathsf{pw}$, and P' is corrupted.
  - This record is $\mathsf{compromised}$.
- If this record is fresh, both parties are honest, and there exists a record $(\mathsf{P}', \mathsf{P}, \mathsf{pw}')$ with $\mathsf{pw}' = \mathsf{pw}$, choose $\mathsf{skey} \leftarrow_{\$} \{0,1\}^{\lambda}$. Output skey to P, and append skey to the record $(\mathsf{P}, \mathsf{P}', \mathsf{pw})$.
- If this record is fresh, both parties are honest, and there exists a record $(\mathsf{P}, \mathsf{P}', \mathsf{pw}', \mathsf{skey})$ with $\mathsf{pw}' = \mathsf{pw}$, output skey to P.
- If none of the above rules apply, choose $\mathsf{skey}' \leftarrow_{\$} \{0,1\}^{\lambda}$ and output it to party P.

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

Fig. 1: The PAKE ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$.

4.3 Subversion-resilient PAKE from M-TSPHFs

In Figure 2, we show our subversion-resilient PAKE protocol from M-TSPHF. In what follows, Enc is the encryption algorithm of a rerandomizable CPA-secure encryption scheme; $\mathcal{P}, \mathcal{V}$, and ZKEval are algorithms of a CM-NIZK for the language $\mathcal{L}_{\mathsf{lpp}} = \{c \mid \exists(\mathsf{aux}, w) \text{ s.t. } \mathsf{Enc}(\mathsf{lpp}, \mathsf{aux}; w) = c\}$ with a set of allowable transformations $\mathcal{T} = \{T_{\widetilde{w}}\}$ mapping a ciphertext c encrypting a message aux with randomness w into a new statement $\widetilde{c}$ obtained from rerandomizing c with randomness $\widetilde{w}$. Finally, the remaining algorithms are part of a M-TSPHF for $L_{\mathsf{lpp},\mathsf{aux}} = \{c \mid \exists w \text{ s.t. } \mathsf{Enc}(\mathsf{lpp}, \mathsf{aux}; w) = c\}$. We implicitly assume that the public parameters pp include both the linear public parameters lpp required from the encryption scheme and for the M-TSPHF, as well as the public parameters required for the CM-NIZK.

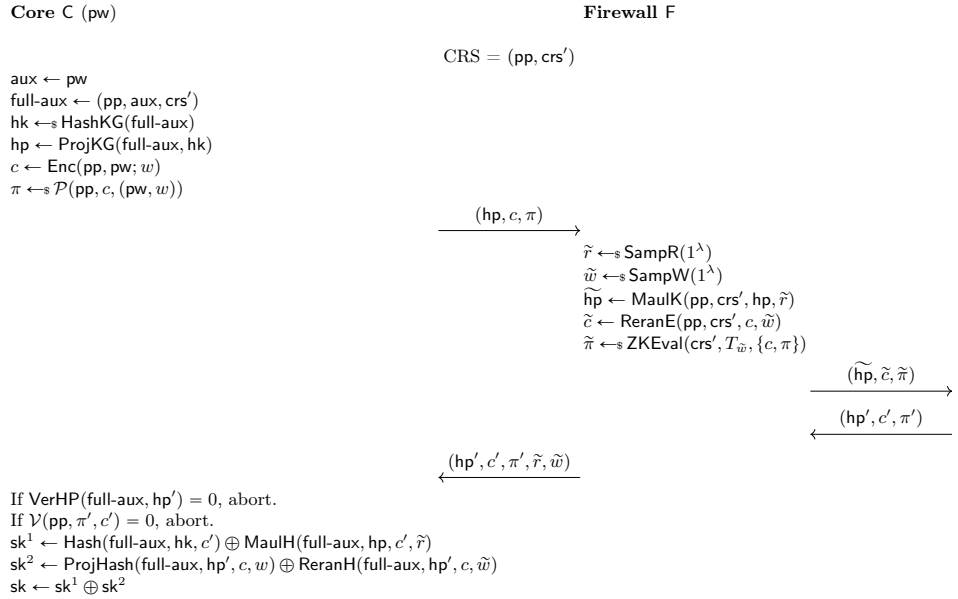

Fig. 2: Our subversion-resilient PAKE protocol from M-TSPHFs.

Theorem 2. *The protocol depicted in Figure 2 wsrUC-realizes the $\mathcal{F}_{\mathsf{PAKE}}$ ideal functionality in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model, assuming secure channels between cores and their respective firewall, a rerandomizable CPA-secure PKE scheme, an M-TSPHF, and a CM-SSE NIZK with straight-line extraction.*

Lemma 1. *The firewalls in Figure 2 are strongly sanitizing.*

We defer the proofs of Theorem 2 and Lemma 1 to Appendix E.

Lemma 2. *For every adversary $\mathcal{A}$ corrupting either party maliciously in an execution of protocol Π as in Figure 2 in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model, there exists a simulator $\mathcal{S}$ such that, for all environments $\mathcal{E}$, the following holds:*

$$\mathtt{Exec}^{\mathcal{F}_{\mathsf{CRS}}}_{\Pi,\mathcal{A},\mathcal{E}} \approx \mathtt{Exec}_{\mathsf{Wrap}(\mathcal{F}_{\mathsf{PAKE}}),\mathcal{S},\mathcal{E}}$$

Proof. In the following proof, we denote $\mathsf{full\text{-}aux} \leftarrow (\mathsf{pp}, \mathsf{aux}, \mathsf{crs}')$ for compactness of notation. We provide a description of the simulator in what follows.

Description of the simulator. The simulator $\mathcal{S}$ first generates the CRS knowing its trapdoors (i.e., $\mathsf{dk}, \tau_{\mathsf{CM\text{-}NIZK}}, \tau_{\mathsf{T\text{-}MSPHF}}$). Then:

- **Upon receiving** $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j)$ **from** $\mathcal{F}_{\mathsf{PAKE}}$**:**
  $\mathcal{S}$ simulates the behaviour of party P_i. In particular, since F_i is semi-honest, $\mathcal{S}$ should only simulate the behaviour of C_i. For that, it computes a key-pair $(\mathsf{hk}, \mathsf{hp})$ for the M-TSPHF, an encryption c of dummy password 1, and a

proof π on statement c by running the CM-NIZK simulator using trapdoor $\tau_{\mathsf{CM\text{-}NIZK}}$. Finally, since F_i is semi-honest, $\mathcal{S}$ has access to the randomness $\widetilde{w}$ used to rerandomize c to $\widetilde{c}$, and to the randomness $\widetilde{r}$ used to rerandomize hp to $\widetilde{\mathsf{hp}}$. In the remainder of the proof, these random strings will be omitted while referring to inbound messages for compactness of representation.

– **Upon receiving** (hp', c', π') **from** $\mathcal{A}$ **with destination honest party** P_i**:** $\mathcal{S}$ checks whether $\mathcal{V}(\mathsf{crs}, c', \pi') = 1$. If so, it checks whether (hp', c') match a tuple $(\widetilde{\mathsf{hp}}, \widetilde{c})$ that was output by $\mathcal{S}$ on behalf of uncorrupted C_j and rerandomized by semi-honest F_j. Then, it proceeds as follows:
  - If the above check succeeds, the tuple is deemed to be "oracle-generated", meaning that the adversary did allow uncorrupted P_i to receive the protocol messages originating from uncorrupted P_j. In this case, $\mathcal{S}$ sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$. Since no $\mathsf{TestPwd}$ queries were performed on this record, the dummy P_i will receive a random session key if it terminates before P_j or if the input passwords differed; otherwise, the same session key as the one sent to P_j.
  - If the above check fails, the tuple is deemed to be "non-oracle-generated", meaning that the adversary has actively interfered with the transmission of messages among uncorrupted parties. In this case, $\mathcal{S}$ uses $\tau_{\mathsf{CM\text{-}NIZK}}$ to extract from π' the tuple $((\mathsf{pw}', w'), c'', T)$. Then:
    * If $(\mathsf{pw}', w') = \perp$ and $(c'', T) \neq \perp$, it means $\mathcal{A}$ rerandomized a simulated proof originally generated by $\mathcal{S}$. In this case, even though $\mathcal{A}$ exhibits an accepting proof for a statement c', it does not prove knowledge of the password or the randomness used to generate c' (and hence, it cannot derive the same session key as the target uncorrupted P_i as per the protocol description). For that, $\mathcal{S}$ sends queries $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}_i, \perp)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$, which marks the record as "interrupted" and makes P_i receive a random session key.
    * If $(\mathsf{pw}', w') \neq \perp$, it means $\mathcal{A}$ produced a fresh accepting proof on statement c'. In this case, $\mathcal{S}$ sends query $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}_i, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{PAKE}}$. If the reply is "correct guess", then $\mathcal{S}$ computes the session key as in the real-world protocol run with password pw'. Importantly, $\mathcal{S}$ uses THash with trapdoor $\tau_{\mathsf{T\text{-}MSPHF}}$ to evaluate H_1 instead of $\mathsf{ProjHash}$. This is to ensure H_1 matches with the counterpart computed by $\mathcal{A}$, given that c was originally an encryption of 1 instead of pw'. In particular, $\mathcal{S}$ computes $\mathsf{aux} \leftarrow \mathsf{pw}'$, $H_1 \leftarrow \mathsf{THash}(\mathsf{full\text{-}aux}, \tau_{\mathsf{T\text{-}MSPHF}}, \mathsf{hp}', c)$, $H_2 \leftarrow \mathsf{ReranH}(\mathsf{full\text{-}aux}, c, \tilde{w})$, $H_1' \leftarrow \mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}, c')$, $H_2' \leftarrow \mathsf{MaulH}(\mathsf{full\text{-}aux}, \mathsf{hp}, \tilde{r})$. Finally, $\mathcal{S}$ computes $\mathsf{sk} \leftarrow H_1 \oplus H_2 \oplus H_1' \oplus H_2'$ and sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}_i, \mathsf{sk})$ to $\mathcal{F}_{\mathsf{PAKE}}$, which makes P_i receive sk. If the reply is "wrong guess", $\mathcal{S}$ sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$, which makes P_i receive a random session key.

We now proceed to incrementally build our simulator via a chain of hybrids, starting from G_0 (the real world) until G_{10}, whose distribution is identical to the

simulator described above. These hybrids follow closely the original proof of [9], adapting them to our modifications. Compared to the unsanitized protocol of [9], introducing CM-NIZKs requires simulating up to q proofs and extracting from up to q proofs (where q is the number of activated parties). Hence, we achieve the same security loss as described in [9], with an additional loss bounded by $q \cdot (\epsilon_{\mathsf{ZK}} + \epsilon_{\mathsf{CM\text{-}SSE}})$, where ϵ_{ZK} is the distinguishing advantage of the adversary against zero-knowledge of the CM-NIZK, and $\epsilon_{\mathsf{CM\text{-}SSE}}$ is the advantage of the adversary in winning the CM-SSE game of the CM-NIZK.

Game G_0. This game matches the real world, where the CRS is generated without knowledge of trapdoors and all flows of honest components are generated by $\mathcal{S}$ as per the protocol description, with $\mathcal{S}$ knowing the inputs of each player. More precisely:

- Upon receiving $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j)$ for a honest P_i, $\mathcal{S}$ runs the protocol as prescribed by generating hashing and projection keys $(\mathsf{hk}, \mathsf{hp})$, a ciphertext c encrypting password pw with randomness w, and a proof π with statement c and witness (pw, w). Then, it sends (hp, c, π) to $\mathcal{A}$.
- Upon receiving (hp', c', π') from $\mathcal{A}$ (supposedly generated by C_j and rerandomized by F_j), $\mathcal{S}$ computes $\mathsf{aux} \leftarrow \mathsf{pw}$, $H_1 \leftarrow \mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r)$, $H_2 \leftarrow \mathsf{ReranH}(\mathsf{full\text{-}aux}, c, \widetilde{w})$, $H_1' \leftarrow \mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}, c')$, $H_2' \leftarrow \mathsf{MaulH}(\mathsf{full\text{-}aux}, \mathsf{hp}, \widetilde{r})$. Finally, it computes $\mathsf{sk} \leftarrow H_1 \oplus H_2 \oplus H_1' \oplus H_2'$ and provides sk to P_i.

Game G_{1a}. $\mathcal{S}$ generates the trapdoor of the CM-NIZK and of the CPA-secure encryption scheme, and it simulates all the proofs output by honest cores. This change is computationally indistinguishable due to zero-knowledge of the CM-NIZK.

Game G_{1b}. Upon receiving (hp', c', π') from $\mathcal{A}$ that is non-oracle generated on behalf of P_j, $\mathcal{S}$ extracts a tuple $((\mathsf{pw}', w'), c'', T)$ from the proof. If any of the following conditions occur, output abort: (i) $(\mathsf{pw}, w') \neq \perp \wedge (c', w') \notin R_{\mathcal{L}_{\mathsf{lpp}}}$, (ii) $(c'', T) \neq (\perp, \perp) \wedge (c'' \notin Q_x \vee c' \neq T_x(c'') \vee T \notin \mathcal{T})$, or (iii) $((\mathsf{pw}', w'), c'', T) = (\perp, \perp, \perp)$. In essence, unless $\mathcal{S}$ aborts, it correctly extracts a password or a previously-queried statement and transformation, enabling the distinguishing of oracle-generated queries from non-oracle generated queries (as well as their respective handling) following the description of the ideal-world simulator. This event occurs with negligible probability due to controlled-malleable simulation sound extractability of the CM-NIZK. Note that, even though the reduction does no longer have access to the simulation trapdoor required to simulate proofs as per G_{1a}, it may query the oracle of the challenger of CM-SSE to obtain simulated proofs.

Game G_2. Upon receiving (hp', c', π') from $\mathcal{A}$ that is either (i) oracle-generated by $\mathcal{S}$ on behalf of P_j with password pw', or (ii) non-oracle generated by $\mathcal{A}$ on behalf of P_j with a proof that extracts to pw', the simulator checks whether $\mathsf{pw} \neq \mathsf{pw}'$. If so, it replaces the quantity $H_1' = \mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}, c')$ with a random value. This variation is computationally indistinguishable due to the smoothness of the M-TSPHF.

Game G_3. Upon receiving (hp', c', π') from $\mathcal{A}$ that is either (i) oracle-generated by $\mathcal{S}$ on behalf of P_j with password pw', or (ii) non-oracle generated by $\mathcal{A}$ on behalf of P_j with a proof that extracts to pw', the simulator checks whether $\mathsf{pw} \neq \mathsf{pw}'$. If so, it replaces the session key sk of party P_i with a random value. This is identically distributed to the previous hybrid, as sk is obtained by combining H_1, H_2, H_1', H_2', and H_1' is random by the previous hybrid.

Game G_4. Upon receiving (hp', c', π') from $\mathcal{A}$ that is either (i) oracle-generated by $\mathcal{S}$ on behalf of P_j with password pw' and hashing key hk', or (ii) non-oracle generated by $\mathcal{A}$ on behalf of P_j with a proof that extracts to pw', $\mathcal{S}$ checks whether $\mathsf{pw} = \mathsf{pw}'$. If so, it redefines the computation of hash values H and H'. First, $\mathcal{S}$ introduces an initially-empty set S_L, and then it evaluates the hash values as follows:

- Upon computation of H' with hashing key hk, check if there exists a record $(\mathsf{aux}, \mathsf{hk}, c'; h)$ in S_L. If so, return h. Otherwise, retrieve the projection key hp associated with hashing key hk, compute $h \leftarrow \mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}, c') \oplus \mathsf{MaulK}(\mathsf{full\text{-}aux}, \mathsf{hp}, \tilde{r})$, and append $(\mathsf{aux}, \mathsf{hk}, c'; h)$ to S_L.
- Upon computation of H with projection key hp', retrieve the hashing key hk' associated with hp'[6], and check if there exists a record $(\mathsf{hk}', \mathsf{aux}, c; h)$ in S_L. If this is the case, return h. Otherwise, compute $h \leftarrow \mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r) \oplus \mathsf{ReranH}(\mathsf{full\text{-}aux}, c, \tilde{w})$ and append $(\mathsf{aux}, \mathsf{hk}', c; h)$ to S_L.

Intuitively, whenever two uncorrupted (simulated) cores mutually receive a tuple originating from their (simulated) peer, the set S_L enforces consistency between the output hashes without explicitly computing Hash or $\mathsf{ProjHash}$ more than once[7]. If the adversary instead guesses the password on behalf of P_j, the set S_L ensures that P_i still computes the session key by running the protocol honestly. This game is identically distributed to the previous one as, whenever $\mathsf{pw} = \mathsf{pw}'$, the outputs of the hashes match by the correctness properties of the M-TSPHF.

Game G_5. Upon receiving (hp', c', π') from $\mathcal{A}$ on behalf of P_j that was oracle-generated by $\mathcal{S}$ on behalf of (simulated) P_j with password pw' and hashing key hk', $\mathcal{S}$ checks whether $\mathsf{pw} = \mathsf{pw}'$. If so, $\mathcal{S}$ replaces the evaluation of H' with a random value whenever S_L does not contain a record for hashing key hk (instead of evaluating $\mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}, c') \oplus \mathsf{MaulK}(\mathsf{full\text{-}aux}, \mathsf{hp}, \tilde{r})$).

To prove the indistinguishability of this game, we show a series of sub-hybrids $G(0)$ to $G(q)$, where in sub-hybrid $G(i)$ $\mathcal{S}$ modifies the evaluation of H' as specified above for all j oracle-generated flows (hp, c', π') received from $\mathcal{A}$ in which parties had the same password, with $j \leqslant i$. In sub-hybrid $G(i)$, we denote the i-th oracle-generated tuple (hp', c', π') as the critical flow, with P_a being the

[6] If hp' is non-oracle generated, set $\mathsf{hk}' \leftarrow \perp$, as the hashing key used by the adversary is unknown to $\mathcal{S}$.

[7] This mechanism is equivalent to algorithms HashL, $\mathsf{ProjHashL}$ used within the original security proof.

originator and P_b the receiver (where a, b are the indexes of $\mathsf{NewSession}$ queries used to initialize players $\mathsf{P}_a, \mathsf{P}_b$). For each sub-hybrid, we further define a series of sub-games G_a to G_e.

- **Game G_a.** In this game, $\mathcal{S}$ guesses the index a identifying the a-th $\mathsf{NewSession}$ query. This succeeds with probability $1/q$, where q is the number of activated players. As in the original proof of [9], if the guess fails, the reduction simply outputs a random guess in the distinguishing game.
- **Game G_b.** In this game, whenever P_a receives its flow, it replaces the evaluation of the h within H as follows. Whenever no record containing hk' exists in S_L, $\mathcal{S}$ computes $h \leftarrow \mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}', c) \oplus \mathsf{ReranH}(\mathsf{full\text{-}aux}, c, \tilde{w})$ (instead of $h \leftarrow \mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r) \oplus \mathsf{ReranH}(\mathsf{full\text{-}aux}, c, \tilde{w})$). This game is perfectly indistinguishable from the previous one by correctness of the M-TSPHF. In particular, $\mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r)$ is replaced with $\mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}', c)$, and in this game we are only considering projection keys that were oracle-generated on a valid ciphertext.
- **Game G_c.** For the a-th $\mathsf{NewSession}$ query, $\mathcal{S}$ computes c as an encryption of dummy password 1 instead of pw. This game is computationally indistinguishable from the previous one due to CPA-security of the encryption scheme. Within the reduction, random coins r used to generate the ciphertext are no longer used for the computation of the hash for this case. Moreover, the reduction can still handle non-oracle generated tuples consistently with the ideal-world simulator by extracting information from the respective proof (rather than relying on the decryption oracle of CCA-security of the original security proof).
- **Game G_d.** In this game, $\mathcal{S}$ guesses the index b identifying the b-th $\mathsf{NewSession}$ query, with P_b being the recipient of the critical flow. When P_b receives its flow, $\mathcal{S}$ computes H as in Game G_4 (i.e., by computing it as a random value instead of $\mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}', c) \oplus \mathsf{MaulK}(\mathsf{full\text{-}aux}, \mathsf{hp}, \tilde{r})$ whenever h is unset). This change is computationally indistinguishable due to smoothness of the M-TSPHF, as c is now an encryption of a dummy password (instead of the real password).
- **Game G_e.** For the a-th $\mathsf{NewSession}$ query, $\mathcal{S}$ reverts the variations introduced in Game G_c, computing c as an encryption of the real password pw instead of the dummy password 1. As for Game G_c, this modification is computationally indistinguishable due to CPA-security of the encryption scheme.
- **Game G_f.** For the a-th $\mathsf{NewSession}$ query, $\mathcal{S}$ reverts the variations introduced in Game G_b, replacing the evaluation of $\mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}', c)$ within h back with $\mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r)$ whenever no record containing hk' exists in S_L. This game is perfectly indistinguishable from the previous one by correctness of the M-TSPHF, as we are only considering projection keys that were oracle-generated on a valid ciphertext.

Game G_6. Upon receiving (hp', c', π') from $\mathcal{A}$ that was oracle-generated by $\mathcal{S}$ on behalf of (simulated) P_j with password pw' and hashing key hk', $\mathcal{S}$ checks whether $\mathsf{pw} = \mathsf{pw}'$. If so, $\mathcal{S}$ chooses sk at random, and it will provide the same key to its peer whenever it receives (hp, c, π) from $\mathcal{A}$ on behalf of P_i.

This game is perfectly indistinguishable from the previous one, as sk is computed by combining H with H', and H is a random value. Moreover, by the computation of the hash defined in Games G_4 and G_5, the set S_L will enforce consistency among these random values.

Game G_7. $\mathcal{S}$ fully generates the CRS by also knowing the trapdoor $\tau_{\mathsf{T\text{-}MSPHF}}$ for the M-TSPHF. As in the original proof, we remark that this cannot be done before, as smoothness of the M-TSPHF is not guaranteed to hold if $\tau_{\mathsf{T\text{-}MSPHF}}$ is known. This game is identically distributed to the previous one.

Game G_8. Upon receiving (hp', c', π') from $\mathcal{A}$ on behalf of P_j that is non-oracle generated for which the proof extracts to pw', $\mathcal{S}$ checks whether $\mathsf{pw} = \mathsf{pw}'$. If so, $\mathcal{S}$ computes H_1 as $\mathsf{THash}(\mathsf{full\text{-}aux}, \tau_{\mathsf{T\text{-}MSPHF}}, \mathsf{hp}', c)$ instead of $\mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r)$. This game is indistinguishable from the previous one due to soundness of the M-TSPHF.

Game G_9. Upon receiving $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j)$ for a honest C_i, $\mathcal{S}$ computes c as an encryption of dummy password 1 instead of pw. This game is computationally indistinguishable from the previous one due to CPA-security of the encryption scheme. As before, the random coins r used to generate the ciphertext are no longer used for the computation of the hash. Moreover, as in Game G_c, the reduction can still handle non-oracle generated tuples consistently with the ideal-world simulator by extracting information from the respective proof (rather than relying on the decryption oracle of CCA-security of the original security proof).

Game G_{10}. In this game, we introduce $\mathcal{F}_{\mathsf{PAKE}}$ and remove access of $\mathcal{S}$ to the input passwords. In particular, $\mathcal{S}$ runs the simulator described above.

4.4 Efficiency analysis of our protocol

Finally, we give an overview of the improvements of our novel protocol compared to the srUC PAKE-from-OT protocol of [18]. In particular, we focus on analyzing round complexity and element complexity for λ-bit passwords[8].

It is clear, from Figure 2, that our protocol is a 1-round PAKE. For what concerns element complexity, we require (i) a rerandomizable CPA-secure encryption scheme, (ii) a M-TSPHF, and (iii) a CM-NIZK for the language of rerandomizable CPA-secure encryption schemes. Item (i) can be efficiently instantiated with the ElGamal encryption scheme by using two group elements per party. Item (ii) can be also efficiently instantiated with our construction provided in Section 3. In particular, $\mathsf{hp} = (\gamma, (\chi_1, \chi_2))$, where γ is a single group element and (χ_1, χ_2) are two group elements. Finally, (iii) can be efficiently instantiated (we provide details in Appendix F), requiring 32 group elements.

[8] Alternatively, one may consider n-bit passwords for arbitrary n, and then evaluate the comparison by computing $H(\mathsf{pw})$, where H is a collision-resistant hash function.

Let us now consider the protocol of [18]. Very roughly, this protocol requires two subversion-resilient OTs (one for each party) *for each bit* of the password, as well as establishing a split-authenticated channel between parties. The only known instantiations of these building blocks are formalized in [18] and require, respectively, 6 group elements for each subversion-resilient OT, and a BLS signature for each outbound message (alongside an initialization phase to setup parties' verification keys). From a round-complexity standpoint, the subversion-resilient OTs require two rounds (by batching together all the OT runs), and the split-authenticated channel introduces two additional rounds in its initialization phase, as well as an additional internal round (i.e., a round between a core and its firewall) for each outbound message.

Putting everything together, we have that our subversion-resilient PAKE protocol runs in a single round (as opposed to the 4 rounds and additional internal rounds of [18]) by using 37 group elements per party (as opposed 6λ group elements for a λ-bit password, plus overhead deriving from BLS signatures, for each party). For passwords of sufficient (and realistic) length, we have that our protocol is also more efficient from an element complexity standpoint.

As a baseline, the non-subversion-resilient TSPHF-based PAKE of [9] avoids the sanitation mechanisms introduced here, and therefore has lower constant-factor communication and computation. Our construction adds the cost of projection-key rerandomization, ciphertext rerandomization, and the CM-SSE NIZK. Hence, the price of subversion resilience is a moderate constant-factor overhead, while preserving the optimal one-round complexity.

5 An alternative method for mauling GS proofs

In Appendix F, we show how to instantiate the CM-NIZK used in our PAKE protocol by leveraging the generic construction due to [20]. Crucially, this construction requires a method for mauling Groth-Sahai proofs, which [20] realizes in a generic fashion by considering a set of elementary transformations with respect to which Groth-Sahai proofs are malleable. However, this approach may become cumbersome while dealing with a large number of Groth-Sahai equations, as one may require composing numerous transformations to capture a transformation of interest (in Appendix G, we illustrate a concrete example of this limitation). For this reason, we propose an alternative method for mauling Groth-Sahai proofs. Our method is simpler but has a more restricted, yet still interesting, scope. For the sake of simplicity, in this section we introduce our mauling method only for the case of pairing product equations. In Appendix H, we show that this method is also applicable to multi-exponentiation and arithmetic circuit equations.

Groth-Sahai proofs: a review. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be groups of prime order p and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear pairing. Recall that a pairing product equation consisting of variables $x_1, x_2, \cdots, x_m \in \mathbb{G}_1$ and variables $y_1, y_2, \cdots, y_n \in \mathbb{G}_2$ is

an equation of the following form:

$$\prod_{i=1,j=1}^{i=m,j=n} e(x_i, y_j)^{\gamma_{i,j}} \prod_{i=1}^{m} e(x_i, b_i) \prod_{j=1}^{n} e(a_j, y_j) t = 1 \tag{1}$$

where $\gamma_{i,j} \in \mathbb{Z}_p$, $a_j \in \mathbb{G}_1$, $b_i \in \mathbb{G}_2$ and $t \in \mathbb{G}_T$ are constants. The CRS for the Groth-Sahai proof consists of (u_1, u_2, v_1, v_2) where $u_1, u_2 \in \mathbb{G}_1^2 - (1,1)$ and $v_1, v_2 \in \mathbb{G}_2 - (1,1)$. u_2 (resp. v_2) should be a multiple of u_1 (resp. v_1) when the CRS is in the perfect soundness mode. Here, $-(1,1)$ means that the identity pair $(1,1)$ is excluded from the corresponding set, namely from $\mathbb{G}_1$ for u_1, u_2 and from $\mathbb{G}_2$ for v_1, v_2. The Groth-Sahai proof for Eq. (1), among other elements, contains a set of commitments to x_j's and y_i's computed as follows: $C_i = (1, x_i) u_1^{r_{i1}} u_2^{r_{i2}}$, $D_j = (1, y_j) v_1^{s_{j1}} v_2^{s_{j2}}$ where r_{i1}, r_{i2}, s_{j1} and s_{j2} are sampled uniformly at random from $\mathbb{Z}_p$. The proof also contains other elements. For our purposes, the exact definition of these other elements is not important, but we will denote them with $\pi_1, \pi_2 \in \mathbb{G}_2^2$ and $\psi_1, \psi_2 \in \mathbb{G}_1^2$. The reader may refer to [29] for further details.

For the verification process of the Groth-Sahai proof system, a bilinear mapping $E : \mathbb{G}_1^2 \times \mathbb{G}_2^2 \to \mathbb{G}_t^4$ is defined as follows:

$$E((x_1, x_2), (y_1, y_2)) = (e(x_1, y_1), e(x_1, y_2), e(x_2, y_1), e(x_2, y_2))$$

Having the above notation, the proof can be verified by checking the following equality:

$$\begin{aligned} &\prod_{i=1,j=1}^{i=m,j=n} E(C_i, D_j)^{\gamma_{i,j}} \prod_{i=1}^{m} e(C_i, (1, b_i)) \prod_{j=1}^{n} e((1, a_j), D_j)(1,1,1,t) \\ &= E(u_1, \pi_1) E(u_2, \pi_2) E(\psi_1, v_1) E(\psi_2, v_2), \end{aligned} \tag{2}$$

where the multiplication between $\mathbb{G}_T^4$ elements is component-wise. Here, $(1,1,1,t) \in G_T^4$ denotes the embedding of $t \in G_T$, with 1 denoting the identity element of G_T. When instead of one pairing-product equation, we have multiple pairing product equations, the proof results from batching the proofs for individual equations (without duplicating the commitments to x_i's and y_j's that are shared between equations).

Our proof mauling method. Note the similarity between the LHS of Eq. (1) and the LHS of Eq. (2). The only difference is that x_i's and y_j's are replaced by their commitments and every constant a_j (respectively b_i) is replaced with $(1, a_j)$ (respectively $(1, b_i)$). We call $(1, a_j)$ (respectively $(1, b_i)$) embedding of a_j in $\mathbb{G}_1^2$ (respectively embedding of b_i in $\mathbb{G}_2^2$). We also call $(1,1,1,t)$ embedding of t in $\mathbb{G}_t^4$. By the embedding of a pairing-production equation, we refer to the equation in which e is replaced with E, every constant is replaced with its embedding and every variable over $\mathbb{G}_1$ (respectively $\mathbb{G}_2$) is replaced with a variable over $\mathbb{G}_1^2$ (respectively $\mathbb{G}_2^2$).

Now we use an example to motivate our proof mauling method. In order to show that the example is not just a contrived example, we describe how it arises in context of mauling ElGamal ciphertexts. Let $(A, B) \in \mathbb{G}_1^2$ be an ElGamal ciphertext with $\mathsf{pk} = h_1$. Consider the following system of pairing-product equations:

$$e(g_1, y_1)e(A, g_2^{-1}) = 1, \qquad e(h_1, y_1)e(x_1, g_2)e(B, g_2^{-1}) = 1.$$

One who is in possession of the plaintext m and the randomness r of the encryption can solve the equation by setting $y_1 = g_2^r$ and $x_1 = m$.

Now consider a mauling transformation on the ciphertext that takes (A, B) to $(g_1^\alpha A, h_1^\alpha B)$. We could consider this transformation as a mauling transformation taking the above system of equations to the following system:

$$e(g_1, y_1')e(g_1^\alpha A, g_2^{-1}) = 1, \qquad e(h_1, y_1')e(x_1', g_2)e(h_1^\alpha B, g_2^{-1}) = 1.$$

A feature of this transformation that is of interest to us is that the change to the constants can be cancelled by a corresponding change to the variables. Explicitly, for every x_1 and y_1, if we set $x_1' = x_1$ and $y_1' = y_1 g_2^\alpha$, then:

$$\begin{aligned} e(g_1, y_1')e(g_1^\alpha A, g_2^{-1}) &= e(g_1, y_1)e(A, g_2^{-1}) \\ e(h_1, y_1')e(x_1', g_2)e(h_1^\alpha B, g_2^{-1}) &= e(h_1, y_1)e(x_1, g_2)e(B, g_2^{-1}). \end{aligned}$$

This feature also holds for the embedding of the above equations. Concretely, for every $X_1 \in \mathbb{G}_1^2$ and $Y_1 \in \mathbb{G}_2^2$, if we set $X_1' = X_1$ and $Y_1' = Y_1(1, g_2)^\alpha$, then:

$$\begin{aligned} E((1, g_1), Y_1')E((1, g_1^\alpha A), (1, g_2^{-1})) &= E((1, g_1), Y_1)E((1, A), (1, g_2^{-1})) \\ E((1, h_1), Y_1')E(X_1', (1, g_2))E((1, h_1^\alpha B), (1, g_2^{-1})) &= E((1, h_1), Y_1)E(X_1, (1, g_2))E((1, B), (1, g_2^{-1})). \end{aligned}$$

In the following definition, we formally define this property of the transformation.

Definition 3. *Le $\mathcal{T}$ be a transformation over the class of pairing-product equations. We call T domain-transforming if the change made to the constants by $\mathcal{T}$, can be cancelled by a corresponding efficient change to the variables. More formally, we call T domain-transforming, if there is an efficient function $I_{\mathcal{T}}$ such that for every set of constants $\mathsf{coef} = (\gamma_{i,j}, a_j, b_i, t)_{i=1,j=1}^{i=m,j=n}$, if $\mathcal{T}$ takes coef to $\mathsf{coef}' = (\gamma_{i,j}', a_j', b_i', t')_{i=1,j=1}^{i=m,j=n}$, then the following holds:*

$$\begin{aligned} &\prod_{i=1,j=1}^{i=m,j=n} E(X_i, Y_j)^{\gamma_{i,j}} \prod_{i=1}^{m} E(X_i, (1, b_i)) \prod_{j=1}^{n} E((1, a_j), Y_j))(1, 1, 1, t) \\ &\equiv \prod_{i=1,j=1}^{i=m,j=n} E(X_i', Y_j')^{\gamma_{i,j}'} \prod_{i=1}^{m} E(X_i', (1, b_i')) \prod_{j=1}^{n} E((1, a_j'), Y_j')(1, 1, 1, t') \end{aligned}$$

where $(X_i', Y_j')_{i=1,j=1}^{i=m,j=n} = I_{\mathcal{T}}\big((X_i, Y_j)_{i=1,j=1}^{i=m,j=n}\big)$.

Now to see why this type of transformation is interesting to us, let $\mathcal{T}$ be a domain-transforming transformation and consider again the LHS of Eq. (2). Based on $\mathcal{T}$ being domain-transforming, we have

$$\begin{aligned}&\prod_{i=1,j=1}^{i=m,j=n} E(C_i, D_j)^{\gamma_{i,j}} \prod_{i=1}^{m} E(C_i, (1, b_i)) \prod_{j=1}^{n} E((1, a_j), D_j)(1, 1, 1, t) \\ &= \prod_{i=1,j=1}^{i=m,j=n} E(C'_i, D'_j)^{\gamma'_{i,j}} \prod_{i=1}^{m} E(C'_i, (1, b'_i)) \prod_{j=1}^{n} E((1, a'_j), D'_j)(1, 1, 1, t'),\end{aligned} \tag{3}$$

where $\gamma'_{i,j}$, a'_j, b'_i and t' are the coefficients of the pairing-product equation that results from applying $\mathcal{T}$ to the original equation and $(C'_i, D'_j)_{i=1,j=1}^{i=m,j=n} = I_{\mathcal{T}}\big((C_i, D_j)_{i=1,j=1}^{i=m,j=n}\big)$. This implies that in Eq. (2), the LHS can be replaced with the RHS of Eq. (3), which means that $(C'_i, D'_j, \pi_1, \pi_2, \psi_1, \psi_2)$ is a valid proof for the mauled pairing-product equation. This gives a method for mauling Groth-Sahai proofs based on domain-transforming transformations. In the following construction, we formalize the above mauling method.

Construction 2 *Let $\mathcal{T}$ be a domain-transforming transformation and $I_{\mathcal{T}}$ be the corresponding efficient function, according to the domain-transforming transformation definition. The mauling of a Groth-Sahai proof $(C_i, D_j, \pi_1, \pi_2, \psi_1, \psi_2)_{i=1,j=1}^{i=m,j=n}$ is given by $(C'_i, D'_j, \pi_1, \pi_2, \psi_1, \psi_2)_{i=1,j=1}^{i=m,j=n}$, where $(C'_i, D'_j)_{i=1,j=1}^{i=m,j=n} = I_{\mathcal{T}}\big((C_i, D_j)_{i=1,j=1}^{i=m,j=n}\big)$.*

Theorem 3. *Construction 2 turns a Groth-Sahai NIWI into a NIWI that is malleable with respect to the domain-transforming transformations.*

References

1. Abdolmaleki, B., Asadi, A.R., Asadi, V.R., Köpsell, S., Mohee, B., Roustaeifar, N., Zarezadeh, M.: Veridp: Verifiable differentially private training. Proceedings on Privacy Enhancing Technologies (PETS) (2026)
2. Abdolmaleki, B., Baghery, K., Lipmaa, H., Zajac, M.: A subversion-resistant SNARK. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 3–33. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70700-6_1
3. Abdolmaleki, B., Campanelli, M., Dao, Q., Mohammadi, S., Roustaeifar, N.: Resettable non-interactive zero-knowledge: Attacks and defenses. Cryptology ePrint Archive, Paper 2026/1036 (2026), https://eprint.iacr.org/2026/1036
4. Abdolmaleki, B., Lipmaa, H., Siim, J., Zajac, M.: On QA-NIZK in the BPK model. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part I. LNCS, vol. 12110, pp. 590–620. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45374-9_20
5. Ateniese, G., Magri, B., Venturi, D.: Subversion-resilient signature schemes. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 364–375. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813635
6. Ball, J., Borger, J., Greenwald, G., et al.: Revealed: how us and uk spy agencies defeat internet privacy and security. Know Your Neighborhood (2013)
7. Bellare, M., Fuchsbauer, G., Scafuro, A.: NIZKs with an untrusted CRS: Security in the face of parameter subversion. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 777–804. Springer, Berlin, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53890-6_26
8. Bellare, M., Paterson, K.G., Rogaway, P.: Security of symmetric encryption against mass surveillance. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 1–19. Springer, Berlin, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44371-2_1
9. Benhamouda, F., Pointcheval, D.: Trapdoor smooth projective hash functions. IACR Cryptol. ePrint Arch. **2013**, 341 (2013), https://api.semanticscholar.org/CorpusID:2274217
10. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
11. Canetti, R., Dachman-Soled, D., Vaikuntanathan, V., Wee, H.: Efficient password authenticated key exchange via oblivious transfer. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 449–466. Springer, Berlin, Heidelberg (May 2012). https://doi.org/10.1007/978-3-642-30057-8_27
12. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Berlin, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_2
13. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Berlin, Heidelberg (May 2005). https://doi.org/10.1007/11426639_24
14. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 68–86. Springer, Berlin, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_5

15. Chakraborty, S., Dziembowski, S., Nielsen, J.B.: Reverse firewalls for actively secure MPCs. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 732–762. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_26
16. Chakraborty, S., Ganesh, C., Pancholi, M., Sarkar, P.: Reverse firewalls for adaptively secure MPC without setup. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 335–364. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92075-3_12
17. Chakraborty, S., Ganesh, C., Sarkar, P.: Reverse firewalls for oblivious transfer extension and applications to zero-knowledge. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part I. LNCS, vol. 14004, pp. 239–270. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30545-0_9
18. Chakraborty, S., Magliocco, L., Magri, B., Venturi, D.: Key exchange in the post-snowden era: Universally composable subversion-resilient PAKE. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part V. LNCS, vol. 15488, pp. 101–133. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0935-2_4
19. Chakraborty, S., Magri, B., Nielsen, J.B., Venturi, D.: Universally composable subversion-resilient cryptography. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part I. LNCS, vol. 13275, pp. 272–302. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-06944-4_10
20. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. pp. 281–300. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
21. Chen, R., Mu, Y., Yang, G., Susilo, W., Guo, F., Zhang, M.: Cryptographic reverse firewall via malleable smooth projective hash functions. In: International Conference on the Theory and Application of Cryptology and Information Security (2016), https://api.semanticscholar.org/CorpusID:19462457
22. Degabriele, J.P., Farshim, P., Poettering, B.: A more cautious approach to security against mass surveillance. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 579–598. Springer, Berlin, Heidelberg (Mar 2015). https://doi.org/10.1007/978-3-662-48116-5_28
23. Dodis, Y., Mironov, I., Stephens-Davidowitz, N.: Message transmission with reverse firewalls—secure communication on corrupted machines. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 341–372. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_13
24. Fuchsbauer, G.: Subversion-zero-knowledge SNARKs. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part I. LNCS, vol. 10769, pp. 315–347. Springer, Cham (Mar 2018). https://doi.org/10.1007/978-3-319-76578-5_11
25. Ganesh, C., Magri, B., Venturi, D.: Cryptographic reverse firewalls for interactive proof systems. In: Czumaj, A., Dawar, A., Merelli, E. (eds.) ICALP 2020. LIPIcs, vol. 168, pp. 55:1–55:16. Schloss Dagstuhl (Jul 2020). https://doi.org/10.4230/LIPIcs.ICALP.2020.55
26. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 524–543. Springer, Berlin, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_33, https://eprint.iacr.org/2003/032.ps.gz
27. Ghadafi, E., Smart, N.P., Warinschi, B.: Groth–sahai proofs revisited. In: Nguyen, P.Q., Pointcheval, D. (eds.) Public Key Cryptography – PKC 2010. pp. 177–192. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

28. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987). https://doi.org/10.1145/28395.28420
29. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N. (ed.) Advances in Cryptology – EUROCRYPT 2008. pp. 415–432. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
30. Jutla, C.S., Roy, A.: Relatively-sound nizks and password-based key-exchange. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7293, pp. 485–503. Springer (2012). https://doi.org/10.1007/978-3-642-30057-8_29, https://doi.org/10.1007/978-3-642-30057-8_29
31. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 293–310. Springer, Berlin, Heidelberg (Mar 2011). https://doi.org/10.1007/978-3-642-19571-6_18
32. Kiltz, E., Pan, J., Wee, H.: Structure-preserving signatures from standard assumptions, revisited. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology – CRYPTO 2015. pp. 275–295. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
33. Liu, J., Wang, Y., Chen, R., Huang, X., Su, J., Yung, M.: Universally composable subversion-resilient authenticated key exchange. Cryptology ePrint Archive, Paper 2025/1521 (2025), https://eprint.iacr.org/2025/1521
34. Mironov, I., Stephens-Davidowitz, N.: Cryptographic reverse firewalls. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 657–686. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6_22
35. Young, A., Yung, M.: The dark side of "black-box" cryptography or: Should we trust capstone? In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 89–103. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
36. Young, A., Yung, M.: Kleptography: Using cryptography against cryptography. In: Fumy, W. (ed.) Advances in Cryptology — EUROCRYPT '97. pp. 62–74. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

A Deferred definitions

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

A.1 Original syntax and security properties of MSPHFs

A malleable SPHF, or MSPHF for short, is a tuple of algorithms $(\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}, \mathsf{MaulK}, \mathsf{MaulHK}, \mathsf{MaulH}, \mathsf{ReranE}, \mathsf{ReranH}, \mathsf{SampR}, \mathsf{SampW})$ such that $(\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$ form an SPHF and the new seven algorithms have the following syntax:

- $\mathsf{MaulK}(\mathsf{pp}, \mathsf{hp}, \widetilde{r})$ takes the public parameters, a projection key hp and a randomness $\widetilde{r}$ and outputs a new projection key $\widetilde{\mathsf{hp}}$.
- $\mathsf{MaulH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{r})$ takes as input the public parameters, a projection key hp, a word x, and a randomness $\widetilde{r}$ and outputs a hash value $\widetilde{\mathsf{hv}}$.
- $\mathsf{MaulHK}(\mathsf{pp}, \mathsf{hk}, \widetilde{r})$ takes the public parameters, a hashing key hk and a randomness $\widetilde{r}$ and outputs a new hashing key $\widetilde{\mathsf{hk}}$.
- $\mathsf{ReranE}(\mathsf{pp}, x, \widetilde{w})$ takes as input the public parameters pp, a word x and a randomness $\widetilde{w}$ and outputs a new word $\widetilde{x}$.
- $\mathsf{ReranH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{w})$ takes the public parameters pp, a projection key hp, a word x and a randomness $\widetilde{w}$ and outputs a hash value $\widetilde{\mathsf{hv}}$.
- $\mathsf{SampR}(\mathsf{pp})$ takes the public parameters pp and outputs a randomness $\widetilde{r}$.
- $\mathsf{SampW}(\mathsf{pp})$ takes the public parameters pp and outputs a randomness $\widetilde{w}$.

In what follows, we summarize the security properties of MSPHFs. It is assumed that there is an operation $*$ over the space of hash values, and that all algorithms have access to pp.

- **Key Indistinguishability**. For any PPT adversary $\mathcal{A}$:

$$\Pr\left[b = b' : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{hp}_1, \mathsf{hp}_2) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b \leftarrow \{0,1\} \\ \widetilde{r} \leftarrow \mathsf{SampR}() \\ \widetilde{\mathsf{hp}} \leftarrow \mathsf{MaulK}(\mathsf{hp}_b, \widetilde{r}) \\ b' \leftarrow \mathcal{A}(\widetilde{\mathsf{hp}}) \end{array}\right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda) \tag{4}$$

- **Projection Consistency**. For every $x \in \mathsf{Set}$:

$$\Pr\left[\mathsf{Hash}(\widetilde{\mathsf{hk}}, x) \neq \mathsf{Hash}(\mathsf{hk}, x) * \widetilde{\mathsf{hv}} : \begin{array}{l} \mathsf{pp}, \tau \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{hk} \leftarrow \mathsf{HashKG}() \\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{aux}, \mathsf{hk}) \\ \widetilde{r} \leftarrow \mathsf{SampR}() \\ \widetilde{\mathsf{hp}} \leftarrow \mathsf{MaulK}(\mathsf{hp}, \widetilde{r}) \\ \widetilde{\mathsf{hv}} = \mathsf{MaulH}(\mathsf{hp}, x, \widetilde{r}) \\ \widetilde{\mathsf{hk}} = \mathsf{MaulHK}(\mathsf{hk}, \widetilde{r}) \end{array}\right] \leqslant \mathsf{negl}(\lambda) \tag{5}$$

– **Element Indistinguishability**. For any PPT adversary $\mathcal{A}$:

$$\Pr\left[b = b' : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (x_1, x_2) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b \leftarrow \{0,1\} \\ \widetilde{w} \leftarrow \mathsf{SampW}() \\ \widetilde{x} \leftarrow \mathsf{ReranE}(x_b, \widetilde{w}) \\ b' \leftarrow \mathcal{A}(\widetilde{x}) \end{array}\right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda) \tag{6}$$

– **Rerandomization Consistency**. For every $x \in \mathsf{Set}$ and aux:

$$\Pr\left[\mathsf{Hash}(\mathsf{hk}, \widetilde{x}) \neq \mathsf{Hash}(\mathsf{hk}, x) * \widetilde{\mathsf{hv}} : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{hk} \leftarrow \mathsf{HashKG}() \\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{aux}, \mathsf{hk}) \\ \widetilde{w} \leftarrow \mathsf{SampW}() \\ \widetilde{x} \leftarrow \mathsf{ReranE}(x, \widetilde{w}) \\ \widetilde{\mathsf{hv}} = \mathsf{ReranH}(\mathsf{hp}, x, \widetilde{w}) \end{array}\right] \leqslant \mathsf{negl}(\lambda) \tag{7}$$

– **Membership Preservation.** For every $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, every $x \in \mathsf{Set}$, every aux and every $\widetilde{r}$ output by SampR, we have $x \in L_{\mathsf{lpp},\mathsf{aux}} \iff \mathsf{ReranE}(x, \widetilde{r}) \in L_{\mathsf{lpp},\mathsf{aux}}$.

A.2 Proof systems

Let $\mathit{\Pi} = (\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ be a tuple of algorithms, for which we consider the following properties:

– **Completeness:** For every $(x, w) \in R$,

$$\mathbb{P}\left[\mathcal{V}(\mathsf{crs}, x, \pi) = 1 : \begin{array}{l} \sigma \leftarrow\!\!\$\, \mathsf{CRSSetup}(1^\lambda) \\ \pi \leftarrow\!\!\$\, \mathcal{P}(\sigma, x, w) \end{array}\right] \geqslant 1 - \mathsf{negl}(\lambda)$$

– **Soundness:** For every PPT adversary $\mathcal{A}$:

$$\mathbb{P}\left[\mathcal{V}(\sigma, x, \pi) = 1 \wedge x \notin L : \begin{array}{l} \sigma \leftarrow\!\!\$\, \mathsf{CRSSetup}(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}(\sigma) \end{array}\right] \leqslant \mathsf{negl}(\lambda)$$

– **Knowledge soundness:** A proof system satisfies knowledge soundness if there exist a PPT algorithm $\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1)$ such that:
  • For every PPT adversary $\mathcal{A}$:

$$\mathbb{P}\left[b = b' : \begin{array}{l} b \leftarrow \{0,1\} \\ \text{if } b = 0 \\ \quad (\sigma, \xi) \leftarrow\!\!\$\, \mathcal{E}_0(1^\lambda) \\ \text{else} \\ \quad \sigma \leftarrow\!\!\$\, \mathsf{CRSSetup}(1^\lambda) \\ b' \leftarrow \mathcal{A}(\sigma) \end{array}\right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda)$$

- For every PPT adversary $\mathcal{A}$:

$$\mathbb{P}\left[\mathcal{V}(\sigma_{\mathsf{ext}}, x, \pi) = 1 \wedge (x, w) \notin R : \begin{array}{l}(\sigma_{\mathsf{ext}}, \xi) \leftarrow_{\$} \mathcal{E}_0(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}(\sigma_{\mathsf{ext}}, \xi) \\ w \leftarrow \mathcal{E}_1(\sigma_{\mathsf{ext}}, \xi, x, \pi)\end{array}\right] \leqslant \mathsf{negl}(\lambda)$$

– **Witness indistinguishability:** A proof system satisfies witness indistinguishability if there exists a PPT algorithm WICRS such that:

- For every PPT adversary $\mathcal{A}$:

$$\mathbb{P}\left[b = b' : \begin{array}{l} b \leftarrow_{\$} \{0,1\} \\ \text{if } b = 0 \\ \quad \sigma \leftarrow_{\$} \mathsf{WICRS}(1^\lambda) \\ \text{else} \\ \quad \sigma \leftarrow_{\$} \mathsf{CRSSetup}(1^\lambda) \\ b' \leftarrow \mathcal{A}(\sigma)\end{array}\right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda)$$

- For every PPT adversary $\mathcal{A}$:

$$\mathbb{P}\left[\begin{array}{c} b = b' \\ \wedge \\ (x_1, w_1) \in R \\ \wedge \\ (x_2, w_2) \in R \end{array} : \begin{array}{l} \sigma \leftarrow_{\$} \mathsf{WICRS}(1^\lambda) \\ ((x_1, w_1), (x_2, w_2)) \leftarrow \mathcal{A}(\sigma) \\ b \leftarrow_{\$} \{0,1\} \\ \pi \leftarrow_{\$} \mathcal{P}(\sigma, x_b, w_b) \\ b' \leftarrow \mathcal{A}(\pi)\end{array}\right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda)$$

– **Zero-knowledge:** A proof system satisfies zero-knowledge if there exist PPT algorithms $\mathcal{S}_0$ and $\mathcal{S}_1$ with the following properties.

- For every PPT adversary $\mathcal{A}$:

$$\mathbb{P}\left[b = b' : \begin{array}{l} b \leftarrow_{\$} \{0,1\} \\ \text{if } b = 0 \\ \quad (\sigma, \tau) \leftarrow_{\$} \mathcal{S}_0(1^\lambda) \\ \text{else} \\ \quad \sigma \leftarrow_{\$} \mathsf{CRSSetup}(1^\lambda) \\ b' \leftarrow \mathcal{A}(\sigma)\end{array}\right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda)$$

- For every PPT adversary $\mathcal{A}$:

$$\mathbb{P}\left[\begin{array}{c} b = b' \\ \wedge \\ (x, w) \in R \end{array} : \begin{array}{l} (\sigma_{\mathsf{sim}}, \tau_{\mathsf{sim}}) \leftarrow_{\$} \mathcal{S}_0(1^\lambda) \\ (x, w) \leftarrow \mathcal{A}(\sigma_{\mathsf{sim}}, \tau_{\mathsf{sim}}) \\ b \leftarrow_{\$} \{0,1\} \\ \text{if } b = 0 \\ \quad \pi \leftarrow_{\$} \mathcal{P}(\sigma_{\mathsf{sim}}, x, w) \\ \text{else} \\ \quad \pi \leftarrow_{\$} \mathcal{S}_1(\sigma_{\mathsf{sim}}, \tau_{\mathsf{sim}}, x) \\ b' \leftarrow \mathcal{A}(\pi)\end{array}\right] \leqslant \frac{1}{2} + \mathsf{negl}(\lambda)$$

– **Rerandomizability:** A proof system is said to be *rerandomizable* if there exists an algorithm Rerand such that, for every PPT adversary $\mathcal{A}$, the following holds:

$$\mathbb{P}\left[\begin{array}{c|l} \mathcal{V}(\mathsf{crs},x,\pi)=1 \\ \wedge\ b=b' \end{array}\right.\left.\begin{array}{l} \mathsf{crs}\leftarrow\mathsf{CRSSetup}(1^\lambda),\ (x,w,\pi)\leftarrow\mathcal{A}(\mathsf{crs}) \\ b\leftarrow_\$\{0,1\} \\ \widetilde{\pi}\leftarrow_\$ \begin{cases}\mathsf{Rerand}(\mathsf{crs},x,\pi) & \text{if } b=0\\ \mathcal{P}(\mathsf{crs},x,w) & \text{if } b=1\end{cases} \\ b'\leftarrow\mathcal{A}(\widetilde{\pi}) \end{array}\right] \leqslant \frac{1}{2}+\mathsf{negl}(\lambda)$$

B Proof of Construction 1

Theorem 4. *Construction 1 is a M-TSPHF for the language of ElGamal ciphertexts.*

Proof. The TSPHF part is as in [9]. We only prove the malleability part.

Projection key consistency follows from the following:

$$e((\gamma g_0^{\widetilde{\alpha_1}} h_0^{\widetilde{\alpha_2}})^w, g_1) = e(\gamma^w, g_1) e(\mathsf{ct}_1^{\widetilde{\alpha_1}} (\mathsf{ct}_2/\mathsf{aux})^{\widetilde{\alpha_1}}, g_1).$$

Projection key indistinguishability. In the projection key indistinguishability game, let $\mathsf{hp}_b = (\gamma, \chi_1, \chi_2)$ be the projection key chosen by the challenger for mauling. If hp_b verifies, then the distribution of $\widetilde{\mathsf{hp}_b} = (\gamma g_0^{\widetilde{\alpha_1}} h_0^{\widetilde{\alpha_2}}, \chi_1 g_1^{\tau' \widetilde{\alpha_1}}, \chi_2 g_1^{\tau' \widetilde{\alpha_2}})$ is independent of hp_b, because $\chi_1 g_1^{\tau' \widetilde{\alpha_1}}$ and $\chi_2 g_1^{\tau' \widetilde{\alpha_2}}$ are distributed uniformly and given these two elements, $\gamma g_0^{\widetilde{\alpha_1}} h_0^{\widetilde{\alpha_2}}$ is the unique element that satisfies $e(x, g_1) = e(g_0, \chi_1 g_1^{\tau' \widetilde{\alpha_1}}) e(h_0, \chi_2 g_1^{\tau' \widetilde{\alpha_2}})$.

Element consistency is proved by the following equation:

$$e((\mathsf{ct}_1 g_0^{\tilde{w}})^{\alpha_1} (\mathsf{ct}_2 h_0^{\tilde{w}}/\mathsf{aux})^{\alpha_2}, g_1) = e(\mathsf{ct}_1^{\alpha_1} (\mathsf{ct}_2/\mathsf{aux})^{\alpha_2}, g_1) e((g_0^{\alpha_1} h_0^{\alpha_2})^{\tilde{w}}, g_1)$$

Element indistinguishability is proved by a hybrid argument. Let H_0 be the element indistinguishability game where the adversary provides the challenger with (ct^1, ct^2) and the challenge rerandomized ciphertext is ct^b where $\mathsf{ct}^b = (g_0^{\tilde{w}} \mathsf{ct}_1^b, h_0^{\tilde{w}} \mathsf{ct}_2^b)$. In H_1, the challenge rerandomized ciphertext is $(g_0^{\tilde{w}} \mathsf{ct}_1^b, U \mathsf{ct}_2^b)$ where U is a uniform and independent element from $\mathbb{G}_0$. Indistinguishability of H_0 and H_1 follows from the DDH assumption in $\mathbb{G}_0$. In H_2 the rerandomized challenge ciphertext is (U_1, U_2) where U_1 and U_2 are uniform and independent elements from $\mathbb{G}_0$. H_1 and H_2 are identically distributed. In H_2 adversary's advantage is zero because the rerandomized challenge ciphertext is independent from (ct^1, ct^2).

C Correctness of the srUC PAKE protocol

In what follows, we explicitly prove correctness of the protocol in Figure 2.

Lemma 3. *The protocol in Figure 2 is correct.*

Proof. Let us denote $\mathsf{full\text{-}aux}_i = (\mathsf{pp}, \mathsf{aux}_i, \mathsf{crs}')$. Given that $\mathsf{pw}_i = \mathsf{pw}_{1-i}$, $\mathsf{full\text{-}aux}_i = \mathsf{full\text{-}aux}_{1-i}$; hence, we omit $\mathsf{full\text{-}aux}$ from all algorithm calls for compactness of notation. We analyze the two portions of the key $\mathsf{sk}^1, \mathsf{sk}^2$ separately:

$$
\begin{aligned}
\mathsf{sk}_i^1 &= \mathsf{Hash}(\mathsf{hk}_i, \widetilde{c}_{1-i}) \oplus \mathsf{MaulH}(\mathsf{hp}_i, \widetilde{r}_i) \\
&= \mathsf{Hash}(\widetilde{\mathsf{hk}}_i, \widetilde{c}_{1-i}) && \text{[Projection Consistency]} \\
&= \mathsf{Hash}(\widetilde{\mathsf{hk}}_i, c_{1-i}) \oplus \mathsf{ReranH}(c_{1-i}, \widetilde{w}_{1-i}) && \text{[Rerandomization Consistency]} \\
&= \mathsf{ProjHash}(\widetilde{\mathsf{hp}}_i, c_{1-i}, w_{1-i}) \oplus \mathsf{ReranH}(c_{1-i}, \widetilde{w}_{1-i}) && \text{[Correctness of SPHF]} \\
&= \mathsf{sk}_{1-i}^2
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{sk}_i^2 &= \mathsf{ProjHash}(\widetilde{\mathsf{hp}}_{1-i}, c_i, w_i) \oplus \mathsf{ReranH}(c_i, \widetilde{w}_i) \\
&= \mathsf{Hash}(\widetilde{\mathsf{hk}}_{1-i}, c_i) \oplus \mathsf{ReranH}(c_i, \widetilde{w}_i) && \text{[Correctness of SPHF]} \\
&= \mathsf{Hash}(\widetilde{\mathsf{hk}}_{1-i}, \widetilde{c}_i) && \text{[Rerandomization Consistency]} \\
&= \mathsf{Hash}(\mathsf{hk}_{1-i}, \widetilde{c}_i) \oplus \mathsf{MaulH}(\mathsf{hp}_{1-i}, \widetilde{r}_{1-i}) && \text{[Projection Consistency]} \\
&= \mathsf{sk}_{1-i}^1
\end{aligned}
$$

We have that $\mathsf{sk}_i^1 \oplus \mathsf{sk}_i^2 = \mathsf{sk}_{1-i}^2 \oplus \mathsf{sk}_{1-i}^1$, as desired.

D MTSPHF for ElGamal based on original definition of malleability

We detailed the obstacles of constructing MSPHF and MTSPHF based on the original definition of malleability, in the technical overview. Next we introduced a new definition whose only difference with the original definition is that it gives aux to MaulH and ReranH, unlike the original one. This seemingly small difference may affect the construction and applicability. We have developed a generic and efficient MTSPHF based on the new definition in the main body of the paper. Because the original definition has potentially broader applicability, here we take the effort to elaborate on the original definition and construct an MTSPHF for ElGamal based on it. The latter construction is less efficient and ad-hoc.

In Section D.1, we provide a malleable SPHF for the language of ElGamal ciphertexts, the main ideas of which will be reused in the following subsection to instantiate our M-TSPHF. In Section D.2, we give a M-TSPHF construction for the language of ElGamal ciphertext that relies on a NIZKPoK with special properties. In Section D.3 we give an instantiation for the required NIZKPoK using Groth-Sahai proofs, our mauling method and [20]'s proof rerandomization method.

D.1 Malleable SPHF for ElGamal

If we attempt to start with the classic SPHF for the language of ElGamal ciphertexts, in which $\mathsf{ProjKG}(\mathsf{hk}, \perp) = g_0^{\alpha_1} h^{\alpha_2}$, and try to make it malleable in a similar manner to Construction 1, we observe that aux (which is the plaintext) is required to compute MaulH to preserve correctness, because we would want $\mathsf{MaulH}(\mathsf{hp}, C = (c_1, c_2), \tilde{r} = (\widetilde{\alpha_1}, \widetilde{\alpha_2}))$ to be equal to $c_1^{\widetilde{\alpha_1}} (c_2/\mathsf{aux})^{\widetilde{\alpha_2}}$. But this does not fit the syntax of malleable SPHFs, as MaulH does not take aux.

In the following, we modify the above approach to get a malleable SPHF for the language of ElGamal ciphertexts that fits the syntax flawlessly. The key idea is that we do not need a pair of random elements $(\widetilde{\alpha_1}, \widetilde{\alpha_2})$ to maul hp to $\tilde{\mathsf{hp}}$; rather, a single random element will suffice.

Construction 3 *Assume that group $\mathbb{G}_0$ of prime order p is given and g_0 is a generator for $\mathbb{G}_0$.*

- ***SPHF** part:*
  - $\mathsf{Setup}(1^\lambda)$ *chooses random* $s \leftarrow \mathbb{Z}_p$*, sets* $h = g_0^s$ *and outputs* $\mathsf{pp} = \mathsf{lpp} = h$.
  - $\mathsf{HashKG}()$ *samples* $\mathsf{hk} = (\alpha_1, \alpha_2) \leftarrow \mathbb{Z}_p^2$.
  - $\mathsf{ProjKG}(\mathsf{aux}, \mathsf{hk}) = \mathsf{hp} = g^{\alpha_1} h^{\alpha_2}$
  - $\mathsf{Hash}(\mathsf{aux}, \mathsf{hk}, \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)) = \mathsf{ct}_1^{\alpha_1} (\mathsf{ct}_2/\mathsf{aux})^{\alpha_2}$.
  - $\mathsf{ProjHash}(\mathsf{aux}, \mathsf{hp}, \mathsf{ct}, w)$ *outputs* hp^w.
- ***Malleability** part:*
  - $\mathsf{SampR}()$ *samples* $\tilde{r} = \tilde{\alpha_1} \leftarrow \mathbb{Z}_p$.
  - $\mathsf{MaulK}(\mathsf{hp}, \tilde{r})$ *outputs* $g^{\tilde{\alpha_1}} \mathsf{hp}$
  - $\mathsf{MaulH}(\mathsf{hp}, \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2), \tilde{r}) = \mathsf{ct}_1^{\tilde{\alpha_1}}$.
  - $\mathsf{SampW}() = \tilde{w} \leftarrow \mathbb{Z}_p$.
  - $\mathsf{ReranE}(\mathsf{ct}, \tilde{w}) = \tilde{\mathsf{ct}} = (\mathsf{ct}_1 g^{\tilde{w}}, \mathsf{ct}_2 h^{\tilde{w}})$.
  - $\mathsf{ReranH}(\mathsf{hp}, \mathsf{ct}, \tilde{w})$ *outputs* $\mathsf{hp}^{\tilde{w}}$.

Theorem 5. *If the DDH assumption holds in $\mathbb{G}$, Construction 3 is a malleable SPHF for the language of ElGamal ciphertexts.*

Proof. We prove each property separately.

- **Projection key indistinguishability** follows from the fact that $g^{\tilde{\alpha_1}} \mathsf{hp}$ is distributed uniformly over $\mathbb{G}$ for any $\mathsf{hp} \in \mathbb{G}$.
- **Projection key consistency:** $\tilde{\mathsf{hp}}$ is defined as $g^{\tilde{\alpha_1}} \mathsf{hp}$, so $\tilde{\mathsf{hk}} = (\tilde{\alpha_1} + \alpha_1, \alpha_2)$. Therefore,

$$\begin{aligned}\mathsf{Hash}(\tilde{\mathsf{hk}}, \mathsf{aux}, \mathsf{ct}) &= \mathsf{ct}_1^{\alpha_1 + \tilde{\alpha_1}} (\mathsf{ct}_2/\mathsf{aux})^{\alpha_2} = \mathsf{ct}_1^{\tilde{\alpha_1}} \mathsf{Hash}(\mathsf{hk}, \mathsf{aux}, \mathsf{ct}) \\ &= \mathsf{MaulH}(\mathsf{hp}, \mathsf{ct}, \tilde{r}) \mathsf{Hash}(\mathsf{hk}, \mathsf{aux}, \mathsf{ct})\end{aligned}$$

- **Element indistinguishability** follows from the DDH assumption.
- **Element consistency:**

$$\begin{aligned}\mathsf{Hash}(\mathsf{hk}, \mathsf{aux}, \tilde{\mathsf{ct}}) &= \tilde{\mathsf{ct}}_1^{\alpha_1} (\tilde{\mathsf{ct}}_2/\mathsf{aux})^{\alpha_2} = (\mathsf{ct}_1 g^{\tilde{w}})^{\alpha_1} (\mathsf{ct}_2 h^{\tilde{w}}/\mathsf{aux})^{\alpha_2} \\ &= g^{\tilde{w}\alpha_1} h^{\tilde{w}\alpha_2} \mathsf{ct}_1^{\alpha_1} (\mathsf{ct}_2/\mathsf{aux})^{\alpha_2} = (g^{\alpha_1} h^{\alpha_2})^{\tilde{w}} \mathsf{Hash}(\mathsf{hk}, \mathsf{aux}, \mathsf{ct}) \\ &= \mathsf{hp}^{\tilde{w}} \mathsf{Hash}(\mathsf{hk}, \mathsf{aux}, \mathsf{ct})\end{aligned}$$

Even though the idea of using a single random element to maul projection keys gives us a malleable SPHF, the same approach would not work if we instead consider [9]'s trapdoor SPHF (the TSPHF given by Construction 1). In their construction, the TSPHF defines projection keys as $\mathsf{hp} = (g_0^{\alpha_1} h^{\alpha_2}, g_1^{\tau\alpha_1}, g_1^{\tau\alpha_2})$. In order to make it compatible with our malleability properties, we would have to maul all parts of this hp. However, we do not have many options left to perform this mauling: τ is a fixed setup, and we can only attempt to sanitize α_1 and α_2 (e.g., blinding them with some random $\widetilde{\alpha_1}$ and $\widetilde{\alpha_2}$). We have seen that this approach would require the MaulH algorithm to take aux as input, which is unacceptable for our PAKE protocol. In the next subsection, we consider a different TSPHF as our starting point to circumvent this issue.

D.2 Malleable TSPHF for ElGamal

A naive generic way to convert any SPHF into a TSPHF is to use an extractable NIZK proof of knowledge (NIZKPoK for short). Roughly speaking, this is done by adding to hp a NIZKPoK of an hk such that hp is the projection key of hk. But this would be very inefficient: if we use the NIZKPoK of Groth-Sahai, we have to do a bit-by-bit proof, because in the most general case we don't have any assumption about the domain of hk (for example, we don't assume that it belongs to a group), and in the interesting cases that we have seen in the previous sections, hk belongs to the field of exponents, while Groth-Sahai NIZKPoK only allows us to extract group elements.

In [30], the authors show that in this NIZKPoK based generic method, it is possible to avoid doing bit-by-bit NIZK for the SPHF they use. In [9], it is shown that [30]'s method can be extended to any SPHF given by [9]'s generic SPHF framework. The resulting construction, although less efficient than the generic framework given in Construction 1, is still reasonably efficient.

We base our malleable TSPHF on the NIZKPoK-based generic TSPHF of [9]. More concretely, in the following construction, we specify a few properties for the NIZKPoK that enable us to make [9]'s NIZKPoK-based TSPHF malleable.

Below we set notations to refer to the relations whose NIZKPoK we will use in the following construction, and also to refer to the transformations of interest on the relations.

Notation 1 – *Let R_{SPHF} denote the relation of tuples $\big((g_0, h_0, \gamma, g_1), (\chi_1, \chi_2)\big)$ such that $e(g_0, \chi_1)e(h_0, \chi_2)e(\gamma, g_1)^{-1} = 1$. Let L_{SPHF} be the corresponding NP language. In what follows, when g_0, h_0 and g_1 are clear from the context, the statement (g_0, h_0, γ, g_1) is denoted with $\mathsf{statement}_\gamma$.*

– *Let $\mathcal{T}_{\mathsf{SPHFRerand}}$ denote the set of transformations $T_{\widetilde{\alpha}}$ over L_{SPHF} that map a statement like (g_0, h_0, γ, g_1) to $(g_0, h_0, g_0^{\widetilde{\alpha}}\gamma, g_1)$, for $\widetilde{\alpha} \in \mathbb{Z}_p$. Note that if (χ_1, χ_2) is a witness for (g_0, h_0, γ, g_1), then $(\chi_1 g_0^{\widetilde{\alpha}}, \chi_2)$ is a witness for $(g_0, h_0, g_0^{\widetilde{\alpha}}\gamma, g_1)$.*

Construction 4 *Let $\mathbb{G}_0$, $\mathbb{G}_1$ and $\mathbb{G}_T$ be groups of prime order p, g_0 and g_1 be generators for $\mathbb{G}_0$ and $\mathbb{G}_1$, respectively and $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a bilinear pairing. Assume that $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, (\mathcal{E}_1, \mathcal{E}_2), (\mathcal{S}_1, \mathcal{S}_2), \mathsf{MaulP}, \mathsf{Rerand})$ is a*

NIZKPoK for R_{SPHF} that is malleable with respect to $\mathcal{T}_{\mathsf{SPHFRerand}}$, perfectly rerandomizable and perfectly zero-knowledge. We define a malleable TSPHF with the following algorithms:

- ***TSPHF** part:*
  - $\mathsf{Setup}(1^\lambda)$ *samples* $(s_0, s_1) \leftarrow \mathbb{Z}_p$ *and outputs* $\mathsf{pp} = (1^\lambda, h_0 = g_0^{s_0}, h_1 = g_1^{s_1})$.
  - $\mathsf{HashKG}()$ *samples* $(\alpha_1, \alpha_2) \leftarrow \mathbb{Z}_p$ *and outputs* $\mathsf{hk} = (\alpha_1, \alpha_2)$.
  - $\mathsf{TSetup}()$ *samples* $(\sigma_{\mathsf{ext}}, \tau_{\mathsf{ext}}) \leftarrow \mathcal{E}_1(1^\lambda)$ *and outputs* $(\mathsf{crs}' = \sigma_{\mathsf{ext}}, \tau' = \tau_{\mathsf{ext}})$.
  - $\mathsf{ProjKG}(\mathsf{aux}, \mathsf{hk})$ *sets* $\gamma = g_0^{\alpha_1} h_0^{\alpha_2}$*, uses* $g_1^{\alpha_1}$ *and* $g_1^{\alpha_2}$ *as a witness to create a proof for* $\mathsf{statement}_\gamma$ *(by running* $\mathcal{P}$*) and outputs* $\mathsf{hp} = (\gamma, \pi)$.
  - $\mathsf{Hash}(\mathsf{aux}, \mathsf{hk}, \mathsf{ct})$ *parses* $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ *and outputs* $e(\mathsf{ct}_1^{\alpha_1}(\mathsf{ct}_2/\mathsf{aux})^{\alpha_2}, g_1)$.
  - $\mathsf{ProjHash}(\mathsf{aux}, \mathsf{hp}, \mathsf{ct}, w)$ *outputs* $e(\gamma^w, g_1)$.
  - $\mathsf{THash}(\mathsf{aux}, \tau', \mathsf{hp}, \mathsf{ct})$ *runs* $\mathcal{E}_2(\mathsf{crs}', \tau', \mathsf{statement}_\gamma, \pi)$*, to extract a witness* (χ_1, χ_2) *and outputs* $e(\mathsf{ct}_1, \chi_1)e(\mathsf{ct}_2/\mathsf{aux}, \chi_2)$.
  - $\mathsf{VerHP}(\mathsf{aux}, \mathsf{hp})$ *outputs* $\mathcal{V}(\mathsf{crs}', \mathsf{statement}_\gamma, \pi)$.
- ***Malleability** part:*
  - $\mathsf{SampR}()$ *samples* $\widetilde{\alpha_1} \leftarrow \mathbb{Z}_p$ *and outputs* $\tilde{r} = \widetilde{\alpha_1}$
  - $\mathsf{MaulK}(\mathsf{hp}, \tilde{r})$*: Let* $T_{\widetilde{\alpha_1}}$ *be the transformation that takes* $\mathsf{statement}_\gamma$ *to* $\mathsf{statement}_{g_0^{\widetilde{\alpha_1}}\gamma}$*. Run* $\mathsf{MaulP}(\mathsf{crs}', T_{\widetilde{\alpha_1}}, \mathsf{statement}_\gamma, \pi)$ *to get* $\tilde{\pi}'$*. Then run* $\mathsf{Rerand}(\mathsf{crs}', \mathsf{statement}_{g_0^{\widetilde{\alpha_1}}\gamma}, \tilde{\pi}')$ *to get* $\tilde{\pi}$*. Output* $\tilde{\mathsf{hp}} = (g_0^{\widetilde{\alpha_1}}\gamma, \tilde{\pi})$.
  - $\mathsf{MaulH}(\mathsf{hp}, \mathsf{ct}, \tilde{r})$ *outputs* $e(\mathsf{ct}_1^{\widetilde{\alpha_1}}, g_1)$.
  - $\mathsf{SampW}()$ *outputs* $\tilde{w} \leftarrow \mathbb{Z}_p$.
  - $\mathsf{ReranE}(\mathsf{ct}, \tilde{w})$ *outputs* $\tilde{\mathsf{ct}} = (g_0^{\tilde{w}}\mathsf{ct}_1, h_0^{\tilde{w}}\mathsf{ct}_2)$.
  - $\mathsf{ReranH}(\mathsf{hp}, \mathsf{ct}, \tilde{w})$ *outputs* $e(\gamma^{\tilde{w}}, g_1)$.

Theorem 6. *Construction 4 is a malleable TSPHF for the language of ElGamal ciphertexts.*

Correctness is straightforward. We prove the rest of the properties.

To prove **smoothness**, we use a hybrid argument. H_0 is the original computational smoothness game defined in the definition of TSPHF. In H_1, (crs', τ') are generated by running $\mathcal{S}_1(1^\lambda)$ instead of $\mathcal{E}_1(1^\lambda)$. The indistinguishability of H_0 and H_1 follows from the indistinguishability of the common reference strings in the NIZKPoK (We can reduce the task of distinguishing between these two hybrids to the task of distinguishing a crs' that is generated by $\mathcal{S}_1$ from a crs' that is generated by $\mathcal{E}_1$. Because we don't use τ' in these two hybrids, the fact that the reduction does not get the trapdoor, doesn't make a problem). In H_2, the proof in hp is generated by running $\mathcal{S}_2(\mathsf{crs}', \tau', \mathsf{statement})$ instead of $\mathcal{P}(\mathsf{crs}', \mathsf{statement}, (g_1^{\alpha_1}, g_1^{\alpha_2}))$. The indistinguishability of H_1 and H_2 follows from the zero-knowledge property of the NIZKPoK. The reduction generates pp and hk, but queries the zero-knowledge challenger for (crs', τ'). Computes γ on its own and passes $(\mathsf{statement}, g_1^{\gamma_1}, g_1^{\gamma_1})$ to the zero-knowledge challenger to receive π and sets $\mathsf{hp} = (\gamma, \pi)$. The rest of the experiment is executed by the reduction. If the challenger generates a real proof, we are in H_1 and if he simulates a proof we are in H_2.

Now we prove that in H_2, adversary's chance is negligible. To do so, we reduce this to the computational smoothness game of the classic SPHF for the ElGamal ciphertexts (refer to Construction 3, SPHF part). The reduction receives h_0 and s_0 from the challenger, samples $s_1 \leftarrow \mathbb{Z}_p$, sets $h_1 = g_1^{s_1}$, $\mathsf{pp} = (h_0, h_1)$, $\tau = (s_0, s_1)$. Samples crs' and τ' by running $\mathcal{S}_1(1^\lambda)$. Passes $(\mathsf{pp}, \tau, \mathsf{crs}')$ to the adversary. Upon receiving $(\mathsf{ct}, \mathsf{aux}, \mathsf{st})$ from the adversary, passes them to the challenger. Receives γ from the challenger, sets $\pi \leftarrow \mathcal{S}_2(\mathsf{crs}', \tau', \mathsf{statement})$ (γ determines the $\mathsf{statement}$). Upon receiving h from the challenger, sets $H = e(h, g_1)$ and passes $\mathsf{hp} = (\gamma, \pi)$ and H to the adversary. Receives b' from the adversary and passes it to the challenger.

Soundness is proven by reduction to the extractability of the underlying NIZKPoK. The reduction generates (pp, τ) and queries the extractability challenger for (crs', τ'), then passes $(\mathsf{pp}, \tau, \mathsf{crs}')$ to the adversary. Upon receiving $(\mathsf{hp}, \mathsf{aux}, \mathsf{ct}, w)$ from the adversary, the reduction parses $\mathsf{hp} = (\gamma, \pi), \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ and passes $(\mathsf{statement}, \pi)$ to the challenger. $(\mathsf{ct}, w) \in L_{\mathsf{lpp},\mathsf{aux}}$ and $\mathsf{VerHP}(\mathsf{hp}, \mathsf{ct}) = 1$ together with $\mathsf{THash}(\tau', \mathsf{aux}, \mathsf{hp}, \mathsf{ct}) \neq \mathsf{ProjHash}(\mathsf{aux}, \mathsf{hp}, \mathsf{ct}, w)$ imply that $\mathcal{V}(\sigma_{\mathsf{ext}}, \mathsf{statement}, \pi) = 1$ and (χ_1, χ_2) is not a valid witness for $\mathsf{statement}$ (which shows that adversary's chance in the soundness game is no more than the challenger's chance in the extractability game). To see why, note that $\mathsf{VerHP}(\mathsf{hp}, \mathsf{ct}) = 1$ implies that $\mathcal{V}(\sigma_{\mathsf{ext}}, \mathsf{statement}, \pi) = 1$. On the other hand, $(\mathsf{ct}, w) \in L_{\mathsf{lpp},\mathsf{aux}}$ implies that $\mathsf{ct}_1 = g_0^w$ and $\mathsf{ct}_2/\mathsf{aux} = h_0^w$, while $\mathsf{THash}(\tau', \mathsf{aux}, \mathsf{hp}, \mathsf{ct}) \neq \mathsf{ProjHash}(\mathsf{aux}, \mathsf{hp}, \mathsf{ct}, w)$ prove that $e(\mathsf{ct}_1, \chi_1)e(\mathsf{ct}_2/\mathsf{aux}, \chi_2) \neq e(\gamma, g_1)^w$, consequently, $e(g_0, \chi_1)e(h_0, \chi_2) \neq e(\gamma, g_1)$, which says that (χ_1, χ_2) is not a valid witness for $\mathsf{statement}$.

Projection key indistinguishability is proved by a hybrid argument. In this argument, some of the hybrids will not be efficiently samplable because their sampling involves brute-force. But this will not invalidate our proof, because wherever we prove the indistinguishability of such hybrids, the reduction is to a perfect indistinguishability. In particular, the reductions will be to the perfect rerandomizability and to the perfect zero-knowledge. Let H_0 be the original projection key indistinguishability game. In other words, the challenger generates pp and crs', passes them to the adversary and assuming that the adversary provides $\mathsf{hp}_1 = (\gamma_1, \pi_1)$ and $\mathsf{hp}_2 = (\gamma_2, \pi_2)$, the challenger runs $\mathsf{MaulP}(\mathsf{crs}', T_{\widetilde{\alpha_1}}, \mathsf{statement}_{\gamma_b}, \pi_b)$ to get $\widetilde{\pi}'$, then runs $\mathsf{Rerand}(\mathsf{crs}', \mathsf{statement}_{g_0^{\widetilde{\alpha_1}}\gamma_b}, \widetilde{\pi}')$ to get $\widetilde{\pi}$ and passes $(g_0^{\widetilde{\alpha_1}}\gamma_b, \widetilde{\pi})$ to the adversary, as the challenge mauled key.

In H_1, crs' is generated in the perfect rerandomizability mode, instead of extractability mode. Indistinguishability of H_1 and H_0 follows immediately from the definition of extractability.

In H_2 the challenger finds a witness $w_b = (\chi_1, \chi_2)$ for the statement $\mathsf{statement}_{\gamma_b}$, by doing a brute-force search, but ignores the witness. H_2 and H_1 are identically distributed.

In H_3, instead of generating a mauled proof $\widetilde{\pi}'$ and rerandomizing, the challenger generates a fresh proof $\widetilde{\pi} \leftarrow \mathcal{P}(\mathsf{crs}', \mathsf{statement}_{g_0^{\widetilde{\alpha_1}}\gamma_b}, (\chi_1 g_1^{\widetilde{\alpha_1}}, \chi_2))$, using the found witness. Indistinguishability of H_3 and H_2 can be reduced to the perfect

rerandomizability of underlying NIZKPoK. The reduction receives crs' from the rerandomizability challenger, generates pp and passes $(\mathsf{pp}, \mathsf{crs}')$ to the adversary. After receiving $\mathsf{hp}_1 = (\gamma_1, \pi_1)$ and $\mathsf{hp}_2 = (\gamma_2, \pi_2)$ from the adversary, the reduction runs $\mathsf{MaulP}(\mathsf{crs}', T_{\widetilde{\alpha_1}}, \mathsf{statement}_{\gamma_b}, \pi_b)$ to get $\widetilde{\pi}'$. On the other hand, the reduction finds a witness (χ_1, χ_2) for the satisfiability of $\mathsf{statement}_{\gamma_b}$, by doing a brute-force search and passes $(\mathsf{statement}_{g_0^{\widetilde{\alpha_1}}\gamma_b}, (\chi_1 g_1^{\widetilde{\alpha_1}}, \chi_2), \widetilde{\pi}')$ to the rerandomizability challenger. Now if the challenger actually rerandomizes $\widetilde{\pi}'$, we are in H_2, otherwise we are in H_3.

In H_4, crs' is generated in the perfect zero-knowledge mode (the challenger would also have the corresponding trapdoor). Indistinguishability of H_4 and H_3 is immediate based on the zero-knowledge definition.

In H_5, the challenger generates a simulated proof by running $\widetilde{\pi} \leftarrow \mathcal{S}_2(\mathsf{crs}_{\mathsf{sim}}, \tau_{\mathsf{sim}}, \mathsf{statement}_{g_0^{\widetilde{\alpha_1}}\gamma_b})$. The indistinguishability of H_5 and H_4 is reduced to the perfect zero-knowledge of the underlying NIZKPoK. The reduction receives crs' from the zero-knowledge challenger, generates pp and passes $(\mathsf{pp}, \mathsf{crs}')$ to the adversary. After receiving $\mathsf{hp}_1 = (\gamma_1, \pi_1)$ and $\mathsf{hp}_2 = (\gamma_2, \pi_2)$ from the adversary, the reduction finds a witness (χ_1, χ_2) for the satisfiability of $\mathsf{statement}_{\gamma_b}$, by doing a brute-force search and passes $(\mathsf{statement}_{g_0^{\widetilde{\alpha_1}}\gamma_b}, (\chi_1 g_1^{\widetilde{\alpha_1}}, \chi_2))$ to the zero-knowledge challenger. If the challenger generates a real proof we are in H_4, otherwise we are in H_5.

In H_6, the challenge mauled projection key consists of $(g_0^{\widetilde{\alpha_1}}, \widetilde{\pi})$, where $\widetilde{\pi} \leftarrow \mathcal{S}_2(\mathsf{crs}_{\mathsf{sim}}, \tau_{\mathsf{sim}}, \mathsf{statement}_{g_0^{\widetilde{\alpha_1}}})$. H_6 and H_5 are perfectly indistinguishable, because $g_0^{\widetilde{\alpha_1}}$ and $\gamma_b g_0^{\widetilde{\alpha_1}}$ have the same distribution. On the other hand, in H_6, the adversary's chance is zero, because the challenge mauled projection key is independent of the adversary's projection keys.

Projection consistency is straightforward.

Let H_0 be the **element indistinguishability** game where the adversary provides the challenger with (ct^1, ct^2) and the challenge rerandomized ciphertext is ct^b where $\mathsf{ct}^b = (g_0^{\tilde{w}} \mathsf{ct}_1^b, h_0^{\tilde{w}} \mathsf{ct}_2^b)$. In H_1, the challenge rerandomized ciphertext is $(g_0^{\tilde{w}} \mathsf{ct}_1^b, U \mathsf{ct}_2^b)$ where U is a uniform and independent element from $\mathbb{G}_0$. Indistinguishability of H_0 and H_1 follows from the DDH assumption in $\mathbb{G}_0$. In H_2 the rerandomized challenge ciphertext is (U_1, U_2) where U_1 and U_2 are uniform and independent elements from $\mathbb{G}_0$. H_1 and H_2 are identically distributed. In H_2 adversary's advantage is zero because the rerandomized challenge ciphertext is independent from (ct^1, ct^2).

Rerandomization consistency immediately follows from the construction.

D.3 Instantiation of the required NIZKPoK

In this section we combine Groth-Sahai proofs and our proof mauling method and [20]'s rerandomization method to instantiate the NIZK component that is needed in the previous section, namely a NIZKPoK for R_{SPHF} that is malleable with respect to $\mathcal{T}_{\mathsf{SPHFRerand}}$ and has perfect rerandomizability and perfect zero-knowledge.

We start with the following equation:

$$e(g_0, y_1)e(h_0, y_2)e(\gamma, g_1)^{-1} = 1 \tag{8}$$

But in order for the final proof system to have the desired properties, the system of equations should have the following properties:

- It should be **tractable**, so that the proof system is zero-knowledge.
- The corresponding transformations should be **domain transforming**, so that we can make the proof system malleable.

In [29], they introduce a method for making a system of equations tractable. Their method works for our system of equations but from the efficiency viewpoint, it is not the best method as it results in a system with many equations. The approach that we use to make Equation (8) tractable is an adaption of their method to our case that is more efficient and is as follows: First choose a constant from the equation that once replaced with 1, makes the equation hold trivially. In this case γ is such a constant. Then in the original equation, replace this constant with a new variable (x_1 in our case) and add two new equations: $x_1 = \gamma^\theta$ and $\theta = 1$. Now this set of equations is equivalent with the original equation, but when considered individually, the new equations can be satisfied trivially by using different values for θ in different equations. In other words, the new system of equations is tractable. This describes why we have added Equations (9, 10, 11) in the following system of equations. The role of Equation (12) will become clear, shortly.

$$e(g_0, y_1)e(h_0, y_2)e(x_1, g_1)^{-1} = 1 \tag{9}$$

$$x_1^{-1}\gamma^\theta = 1 \tag{10}$$

$$\theta = 1 \tag{11}$$

$$g_0^\theta g_0^{-1} = 1 \tag{12}$$

It can be seen that $\mathcal{T}_{\mathsf{SPHFRerand}}$ is not a domain transforming set of transformations for the above system of equations, and Equation (10) is the source of the problem. To see that $\mathcal{T}_{\mathsf{SPHFRerand}}$ is domain-transforming with respect to the rest of the equations, let $T_{\tilde{\alpha}} \in \mathcal{T}_{\mathsf{SPHFRerand}}$ be the transformation that takes (g_0, h_0, γ, g_1) to $(g_0, h_0, g_0^{\tilde{\alpha}}\gamma, g_1)$. In order to show that $T_{\tilde{\alpha}}$ is domain transforming, informally, we have to show that the change made to the constants by $T_{\tilde{\alpha}}$ can be cancelled by a corresponding change to the variables[9]. By this we mean that after changing the constants in LHS of the equation according to the transformation, we can change the variables in a way that the value of the LHS remains unchanged[10]. But this is trivial, because $T_{\tilde{\alpha}}$ makes no change to the constants

[9] In Section 5, we defined our mauling method only for pairing-product equations. In Appendix H, we generalize the method in a natural way to support other types of equations. Nevertheless these lines can be followed without reading the appendix.

[10] More accurately, we have to show this cancellation for the embedding (as defined in Section 5) of the equation. But because we only multiply the variables by constant factors, the embedding version follows immediately.

in Equations 9, 11 and 12 and we can leave the variables unchanged! If we use this form of changing the variables (no change at all), then the mauling of the proofs corresponding to Equations 9, 11 and 12, would not change the proof elements at all. Among these untouched proof elements is the commitment to x_1, which here we denote by c. The problem with this is that even though the mauled proofs corresponding to different equations are verified independently, the commitments are shared between proofs and should be mauled in a unique way. Especially, because c is shared between the proofs of Equations 9 and 10, we have to make sure that the way we maul c for Equation 9, is consistent with the way we maul it for Equation 10. Therefore, instead of not changing the variables at all, we use the following change of variables:

$$x'_1 = g_0^{\alpha} x_1 \quad , \quad y'_1 = g_1^{\alpha} y_1 \tag{13}$$

and $y'_2 = y_2$. This is another form of changing the variables to satisfy the definition of domain-transforming transformation. In other words:

$$e(g_0, y_1)e(h_0, y_2)e(x_1, g_1)^{-1} = e(g_0, y'_1)e(h_0, y'_2)e(x'_1, g_1)^{-1}.$$

Therefore to maul the components corresponding to 9, 11 and 12, we use the method that we introduced in Section 5, and to maul the components corresponding to 10, we will use an ad-hoc method, and that is why we added 12 that at first seemed to be redundant. The Groth-Sahai proof for this extra equation provides terms that let us do the ad-hoc mauling of the proof corresponding to Equation 10.

Construction 5 *Let* $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ *be the Groth-Sahai proof system for the system of equations 9,10,11 and 12. We make the proof system rerandomizable and malleable with respect to* $\mathcal{T}_{\mathsf{SPHFRerand}}$ *by adding the following algorithms:*

- ***Rerandomization*** *is done by the method given in [20].*
- ***Mauling:*** *The proof components corresponding to 9, 11 and 12 and all of the commitments are mauled according to the algorithm given in Section 5. This is possible because* $\mathcal{T}_{\mathsf{SPHFRerand}}$ *is domain transforming with respect to the aforementioned equations. Mauling of the proof components corresponding to 10 is done as follows: Assuming that the proof component corresponding to 10 is* $(\psi_{2,1}, \pi_{2,1}, \pi_{2,2})$[11] *and the proof component corresponding to 12 is* $\psi_{4,1}$[12]*, define:*

$$\widetilde{\psi_{2,1}} = \psi_{4,1}^{\tilde{\alpha}} \psi_{2,1} \qquad \widetilde{\pi_{2,1}} = \pi_{2,1} \qquad \widetilde{\pi_{2,2}} = \pi_{2,2} \tag{14}$$

Theorem 7. *Construction 5 is a NIZKPoK for the satisfiability of Equation 8 that is malleable with respect to* $\mathcal{T}_{\mathsf{SPHFRerand}}$ *and rerandomizable. It has perfect zero-knowledge and rerandomizability.*

[11] Refer to [29] or Appendix H to see why this proof component consists of 1 element from $\mathbb{G}_1^2$ and 2 elements from $\mathbb{G}_2^2$.

[12] Refer to [29] or Appendix H to see why this proof component consists of 1 element from $\mathbb{G}_1^2$.

Proof. Because the core of the construction is a Groth-Sahai proof, it is a NIZKPoK with perfect zero-knowledge (The CRS is generated in the perfect zero-knowledge mode). Perfect rerandomizability follows from the perfect rerandomizability of Groth-Sahai proofs as shown in [20]. Malleability of the proof components corresponding to 9, 11 and 12 follows from 9. We only need to prove the malleability of the proof components corresponding to 10. Assume that $(\psi_{2,1}, \pi_{2,1}, \pi_{2,2})$ and $\psi_{4,1}$ are valid and c and d are commitments to x_1 and θ, respectively. Then based on the validity of the proof components corresponding to 10 and 12 one has:

$$E(c, v^{-1})E((1, \gamma), d) = E(u_1, \pi_{2,1})E(u_2, \pi_{2,2})E(\psi_{2,1}, v_1) \tag{15}$$

$$E((1, g_0), d)E((1, g_0), v^{-1}) = E(\psi_{4,1}, v_1) \tag{16}$$

Assume that we are mauling the proof for the transformation $T_{\tilde{\alpha}}$ that maps (g_0, h_0, γ, g_1) to $(g_0, h_0, g_0^{\tilde{\alpha}}\gamma, g_1)$. Based on the variable-change in 13 and the way we maul commitments:

$$\widetilde{c} = (1, g_0^{\tilde{\alpha}})c, \quad \widetilde{d} = d \tag{17}$$

Therefore,

$$\begin{aligned}
&E(\widetilde{c}, v^{-1})E((1, g_0^{\tilde{\alpha}}\gamma), \widetilde{d}) = E((1, g_0^{\tilde{\alpha}})c, v^{-1})E((1, g_0^{\tilde{\alpha}}\gamma), d) && \text{[based on 17]}\\
=&E(c, v^{-1})E((1, \gamma), d)E((1, g_0^{\tilde{\alpha}}), v^{-1})E((1, g_0^{\tilde{\alpha}}), d)\\
=&E(u_1, \pi_{2,1})E(u_2, \pi_{2,2})E(\psi_{2,1}, v_1)E(\psi_{4,1}, v_1)^{\tilde{\alpha}} && \text{[based on 15 and 15]}\\
=&E(u_1, \widetilde{\pi_{2,1}})E(u_2, \widetilde{\pi_{2,2}})E(\widetilde{\psi_{2,1}}, v_1) && \text{[based on 14]}
\end{aligned}$$

Which shows that $(\widetilde{\psi_{2,1}}, \widetilde{\pi_{2,1}}, \widetilde{\pi_{2,2}})$ satisfies the verification equation. This completes the proof.

E Omitted Proofs

Theorem 8. *The protocol depicted in Figure 2 wsrUC-realizes the $\mathcal{F}_{\mathsf{PAKE}}$ ideal functionality in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model, assuming secure channels between cores and their respective firewall, a rerandomizable CPA-secure PKE scheme, an M-TSPHF, and a CM-SSE NIZK with straight-line extraction.*

Proof. In order to prove our theorem, we follow the srUC framework by showing (i) correctness of the protocol; (ii) strong sanitation of firewalls, and (iii) simulation of the protocol by assuming honest parties being composed of an Honest core and a SemiHonest RF. (i) trivially follows from the consistency properties of M-TSPHFs. In particular, this is possible given that cores have access to the randomness that was used from their respective firewall to sanitize messages. (ii) follows by showing that firewalls can sanitize outbound messages by leveraging the malleability properties of the respective primitives. Finally, the high-level

idea of item (iii) for the simulation of the protocol consists in adapting the original proof of the non-sanitized protocol of [9] to the modifications we introduced. We summarize the proof strategy of [9] in what follows, and conclude the sketch by outlining how the proof changes in our setting.

Let us consider the simulation of an honest party P within the protocol run. A simulator should run the protocol on behalf of honest parties by computing a ciphertext of some dummy password. Upon receiving a tuple on behalf of the peer P' of P, it first checks whether this tuple was "oracle-generated", where oracle-generated roughly means that the tuple was generated by the simulator itself on behalf of a honest P'. If this is the case, it means the adversary did not actively tamper with the protocol run; therefore, the simulator simply instructs $\mathcal{F}_{\mathsf{PAKE}}$ to output the session keys to the respective dummy parties. Otherwise, the adversary had interfered with the protocol run, and the simulator should ascertain whether the adversary was able to guess the password of party P or not. To do so, it decrypts the non-oracle generated ciphertext by using the trapdoor for the CCA-secure scheme to retrieve the adversary's guess, which is forwarded to $\mathcal{F}_{\mathsf{PAKE}}$ by means of a $\mathsf{TestPwd}$ query. If the guess was correct, the simulator has to "correct" the key generated by P to match the one computed by the adversary. This is achieved by using the trapdoor of the TSPHF (recall that P originally provided an encryption of a dummy password), and then send a NewKey query to $\mathcal{F}_{\mathsf{PAKE}}$ containing the adversarially-generated key. Otherwise, the simulator should just forward a NewKey query to the ideal functionality (which will return random and unrelated password to parties). In order to move from the real world to the simulated world, we run through a series of hybrids in which we first replace session keys for honest parties that receive oracle-generated messages with (possibly related) random keys (i.e., the first case of the aforementioned simulator), and then do the same for non-oracle generated messages (unless the adversary guesses the password; in which case, the session key will be adversarially-controlled).

The most salient variations we introduce compared to the above sketch arise from the replacement of CCA-secure ciphertexts with rerandomizable CPA-secure ones plus CM-NIZKs (particularly, in the characterization of "oracle-generated" tuples, which are now identified according to the output of the extraction from the proofs), as well as the evaluation of portions of session keys due to the malleability properties introduced within the protocol. The latter is mostly a syntactic change. The proof of correctness is deferred to Appendix C. In what follows, we formally strong sanitation and simulatability of the protocol.

Lemma 4. *The firewalls in Figure 2 are strongly sanitizing.*

Proof. In order to prove strong sanitation, we show that no environment $\mathcal{E}$, which allows specious corruptions for the core C but does not corrupt the respective firewall F, is able to distinguish between an execution of the protocol in Figure 2 with an incorruptible C from one that accepts the corruption $\tilde{\mathsf{C}}$. To this extent, we invoke the individual malleability properties of the primitives used for the three types of messages exchanged in the protocol: projection keys, ciphertexts, and NIZKs.

A specious $\tilde{\mathsf{C}}$ might generate the outbound tuple (hp, c, π) in a biased manner. The projection key hp gets mauled to a new key by F with the MaulK algorithm, which outputs a fresh key $\tilde{\mathsf{hp}}$ due to key indistinguishability of the M-TSPHF. The ciphertext c gets rerandomized using the ReranE algorithm, which outputs a fresh ciphertext $\tilde{c}$ by element rerandomizability of the M-TSPHF. The proof π gets mauled to a fresh proof $\tilde{\pi}$ due to strong derivation privacy of the CM-NIZK scheme.

F Instantiation of a CM-NIZK for ElGamal

In this section, we instantiate [20]'s generic CM-SSE-NIZK (which we refer to as CM-NIZK) construction using [32]'s signature scheme and Groth-Sahai NIWI proofs to obtain a CM-NIZK with respect to the class of rerandomization transformations we have discussed in our malleable TSPHF for ElGamal ciphertexts. Together with the aforementioned malleable TSPHF, this CM-NIZK is used in our PAKE protocol.

In this section, assume that groups $\mathbb{G}_0$ and $\mathbb{G}_1$ of prime order p with generators g_0 and g_1, respectively, together with a bilinear pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \leftarrow \mathbb{G}_T$ are given. Denote matrices and vectors with bold letters. For a matrix $\mathbf{A} = [a_{ij}] \in \mathbb{Z}_p^{m \times n}$, denote $[g^{a_{ij}}]$ with $g^{\mathbf{A}}$. For matrices of matching size, $\mathbf{A}$ and $\mathbf{B}$, define $e(g_0^{\mathbf{A}}, g_1^{\mathbf{B}}) = e(g_0, g_1)^{\mathbf{AB}}$. Observe that, given $g_0^{\mathbf{A}}$ and $g_1^{\mathbf{B}}$, we can compute $e(g_0^{\mathbf{A}}, g_1^{\mathbf{B}})$, in particular $\mathbf{A}$ and $\mathbf{B}$ are not required.

The authors of [32] give a structure-preserving signature scheme based on the matrix Diffie–Hellman (MDDH) assumption. Below, we instantiate their construction to obtain a signature scheme for messages containing two group elements based on the decision linear (DLIN) assumption. In this paper we refer to this instantiation as the KPW scheme.

Construction 6 (KPW[32]) *The signature scheme is defined as follows:*

- $\mathsf{SigKGen}(1^\lambda)$ *Samples* $\mathbf{K}$, $\mathbf{K}_0$ *and* $\mathbf{K}_1$ *uniformly from* $\mathbb{Z}_p^{3\times 3}$, *samples* a_1,a_2, b_1 *and* b_2 *uniformly from* $\mathbb{Z}_p$ *and sets* $\mathbf{P}_0 = \mathbf{B}^T\mathbf{K}_0$, $\mathbf{P}_1 = \mathbf{B}^T\mathbf{K}_1$

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ a_1 & 0 \\ 0 & a_2 \end{bmatrix} \quad , \quad \mathbf{B} = \begin{bmatrix} 1 & 1 \\ b_1 & 0 \\ 0 & b_2 \end{bmatrix}.$$

  Public verification-key consists of $\mathbf{vk} = (\mathbf{vk}_1 = g_1^{\mathbf{KA}}, \mathbf{vk}_2 = g_1^{\mathbf{K}_0\mathbf{A}}, \mathbf{vk}_3 = g_1^{\mathbf{K}_1\mathbf{A}}, \mathbf{vk}_4 = g_1^{\mathbf{A}})$. *Secret signing-key is defined as* $\mathsf{sk} = (\mathbf{K}, g_0^{\mathbf{P}_0}, g_0^{\mathbf{P}_1}, g_0^{\mathbf{B}})$.
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$ *takes a message* $\mathsf{msg} = (g_0^{m_1}, g_0^{m_2}) \in \mathbb{G}_0^2$, *samples* $\mathbf{r} \leftarrow \mathbb{Z}_p^2$ *and* $\tau \leftarrow \mathbb{Z}_p$, *sets*
  - $\boldsymbol{\sigma}_1 = g_0^{[1 \ m_1 \ m_2]\mathbf{K} + \mathbf{r}^T(\mathbf{P}_0 + \tau\mathbf{P}_1)}$ *(note that one doesn't require* m_1 *and* m_2 *to compute* σ_1, *but for ease of notation, we make them explicit.)*
  - $\boldsymbol{\sigma}_2 = g_0^{\mathbf{r}^T\mathbf{B}^T}$
  - $\boldsymbol{\sigma}_3 = g_0^{\mathbf{r}^T\mathbf{B}^T\tau}$

- $\sigma_4 = g_1^{\tau}$

and outputs $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3, \sigma_4)$

– $\mathsf{SigVerify}(\mathbf{vk}, \mathsf{msg}, \boldsymbol{\sigma})$ *verifies the following equations:*

$$e(\boldsymbol{\sigma}_1, \mathbf{vk}_4) = e(g_0^{[1 \ m_1 \ m_2]}, \mathbf{vk}_1) e(\boldsymbol{\sigma}_2, \mathbf{vk}_2) e(\boldsymbol{\sigma}_3, \mathbf{vk}_3)$$
$$e(\boldsymbol{\sigma}_2, \sigma_4) = e(\boldsymbol{\sigma}_3, g_1)$$

Notation 2 – *Let* R_{Elgamal} *be the NP relation consisting of tuples of the form* $((\mathsf{ct}, h_0), (m, g_0^w, h_0^w))$*, where* ct *is a valid Elgamal encryption of* m *using public-key* h_0 *and randomness* w*. In other words:*

$$R_{\mathsf{Elgamal}} = \{((g_0^w, m h_0^w, h_0), (m, g_0^w, h_0^w)) | h_0, m \in \mathbb{G}_0, w \in \mathbb{Z}_p\}.$$

Let L_{Elgamal} *be the corresponding NP language.*

– *Let* $\mathcal{T}_{\mathsf{Elgamal}}$ *be the set of rerandomization transformations acting on* R_{Elgamal}*. In other words,* $\mathcal{T}_{\mathsf{Elgamal}}$ *is the set of transformations* $T_{\widetilde{w}}$ *that get a tuple like* $((\mathsf{ct}_1, \mathsf{ct}_2, h_0), (m, u, v))$ *and take it to* $((g_0^{\widetilde{w}} \mathsf{ct}_1, h_0^{\widetilde{w}} \mathsf{ct}_2, h_0), (m, g_0^{\widetilde{w}} u, h_0^{\widetilde{w}} v))$*. Take* $T_{1,\widetilde{w}}$ *and* $T_{2,\widetilde{w}}$ *to be the first and second components of* $T_{\widetilde{w}}$*.*

– *Let* $(\mathsf{SigKGen}, \mathsf{Sign}, \mathsf{SigVerify})$ *be the KPW signature scheme, we take* $R_{\mathsf{Elgamal,KPW}}$ *to be an instantiation of the relation that authors of [20] consider in their CM-SSE-NIZK construction, defined as follows:*

$$R_{\mathsf{Elgamal,KPW}} = \{((X, \mathbf{vk}), (W, g_0^{\widetilde{w}}, h_0^{\widetilde{w}}, X', \boldsymbol{\sigma})) | (X, W) \in R_{\mathsf{Elgamal}} \vee$$
$$(\widetilde{w} \in \mathbb{Z}_p \wedge X' \in L_{\mathsf{Elgamal}} \wedge T_{\widetilde{w}}(X') = X \wedge \mathsf{SigVerify}(\mathbf{vk}, X', \boldsymbol{\sigma}) = 1)\}$$

– *Let* $\mathcal{T}_{\mathsf{Elgamal,KPW}}$ *be the set of transformations over* $R_{\mathsf{Elgamal,KPW}}$*, that [20] consider in their CM-SSE-NIZK construction. Concretely,* $\mathcal{T}_{\mathsf{Elgamal,KPW}}$ *is the set of transformations* $S_{\widetilde{w'}}$ *that take a tuple like* $((X, \mathbf{vk}), (W, g_0^{\widetilde{w}}, h_0^{\widetilde{w}}, X', \boldsymbol{\sigma}))$ *to* $((T_{1,\widetilde{w'}}(X), \mathbf{vk}), (T_{2,\widetilde{w'}}(W), g_0^{\widetilde{w}+\widetilde{w'}}, h_0^{\widetilde{w}+\widetilde{w'}}, X', \boldsymbol{\sigma}))$*.*

Below, we show an instantiation of [20]'s NIZK which is CM-SSE with respect to $\mathcal{T}_{\mathsf{Elgamal}}$, so that it can be used in our PAKE construction.

Construction 7 ([20]) *Let* $(\mathsf{SigKGen}, \mathsf{Sign}, \mathsf{SigKGen})$ *be the KPW signature scheme and* $(\mathsf{CRSSetup}', \mathcal{P}', \mathcal{V}', \mathsf{ZKEval}')$ *be a NIWI proof of knowledge for* $R_{\mathsf{Elgamal,KPW}}$ *that is malleable with respect to* $\mathcal{T}_{\mathsf{Elgamal,KPW}}$*. The CM-NIZK for* $\mathcal{T}_{\mathsf{Elgamal}}$ *defined in [20] works as follows:*

– $\mathsf{CRSSetup}(1^\lambda)$ *outputs* $\mathsf{crs} = (\mathsf{crs}', \mathbf{vk})$ *where* crs' *and* $\mathbf{vk}$ *are generated by* $\mathsf{CRSSetup}'$ *and* $\mathsf{SigKGen}$*, respectively.*

– $\mathcal{P}(\mathsf{crs}, X, W)$ *creates a proof* π *by running* $\mathcal{P}'$ *on* $(X, \mathbf{vk})$ *as statement and* $(W, \perp, \perp, \perp)$ *as witness.*

– $\mathcal{V}(\mathsf{crs}, X, \pi)$ *verifies that* $\mathcal{V}'(\mathsf{crs}', (X, \mathbf{vk}), \pi) = 1$*.*

– $\mathsf{ZKEval}(\mathsf{crs}, T_{\widetilde{w'}}, \pi)$ *outputs* $\mathsf{ZKEval}'(\mathsf{crs}', S_{\widetilde{w'}}, \pi)$*.*

For the above construction, we need a NIWI for $R_{\mathsf{Elgamal,KPW}}$. In order to construct a NIWI for $R_{\mathsf{Elgamal,KPW}}$, we first represent the statement

$\exists(W, g_0^{\widetilde{w}}, h_0^{\widetilde{w}}, X', \boldsymbol{\sigma})$ s.t. $((X, \mathbf{vk}), (W, g_0^{\widetilde{w}}, h_0^{\widetilde{w}}, X', \boldsymbol{\sigma})) \in R_{\mathsf{Elgamal,KPW}}$ in terms of pairing-product and multi-exponentiation equations. Assume that $(X, \mathbf{vk})$ is given. Parse $X = (\mathsf{ct}_1, \mathsf{ct}_2, h_0)$ and $\mathbf{vk} = (\mathbf{vk}_1, \mathbf{vk}_2, \mathbf{vk}_3, \mathbf{vk}_4)$. Observe that a $W = (m, g_0^w, h_0^w)$ for which $(X, W) \in R_{\mathsf{Elgamal}}$ gives a solution to the following system of equations and vice versa ($Z_1 = g_0^w, Z_2 = h_0^w, Z_3 = m, \phi = w$):

$$\begin{aligned} e(\mathsf{ct}_1, g_1)e(x_1, g_1)^{-1} &= 1 \\ e(\mathsf{ct}_2, g_1)e(x_2, g_1)^{-1}e(x_3, g_1)^{-1} &= 1 \\ x_1^{-1}g_0^{\phi} &= 1 \\ x_2^{-1}h_0^{\phi} &= 1 \end{aligned} \tag{18}$$

Also, observe that a $(g_0^{\widetilde{w}}, h_0^{\widetilde{w}}, X', \boldsymbol{\sigma})$ for which $\widetilde{w} \in \mathbb{Z}_p \wedge T_{\widetilde{w}}(X') = X \wedge \mathsf{SigVerify}(\mathbf{vk}, X', \boldsymbol{\sigma}) = 1$, gives a solution to the following system of equations and vice versa ($X' = (x_4, x_5), \theta = \widetilde{w}, x_6 = g_0^{\widetilde{w}}, x_7 = h_0^{\widetilde{w}}, \boldsymbol{\sigma} = (\mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}, x_{11})$)($\mathbf{x}_8$,$\mathbf{x}_9$ and $\mathbf{x}_{10}$ are 1×3 vectors and equations containing them are indeed sets of paring-product equations):

$$\begin{aligned} e(\mathsf{ct}_1, g_1)^{-1}e(x_6, g_1)e(x_4, g_1) &= 1 \\ e(\mathsf{ct}_2, g_1)^{-1}e(x_7, g_1)e(x_5, g_1) &= 1 \\ x_6^{-1}g_0^{\theta} &= 1 \\ x_7^{-1}h_0^{\theta} &= 1 \\ e(\mathbf{x}_8, \mathbf{vk}_4)^{-1}e([g_0\ x_4\ x_5], \mathbf{vk}_1)e(\mathbf{x}_9, \mathbf{vk}_2)e(\mathbf{x}_{10}, \mathbf{vk}_3) &= 1 \\ e(\mathbf{x}_9, x_{11})e(\mathbf{x}_{10}, g_1)^{-1} &= 1 \end{aligned} \tag{19}$$

Now we need to make a disjunctive combination of the above two sets of equations to reach at a single system of equations, a solution of which gives a witness $(W, g_0^{\widetilde{w}}, h_0^{\widetilde{w}}, X', \boldsymbol{\sigma})$ for $(X, \mathbf{vk})$ and vice versa. We use [20]'s technique for this purpose (We don't use the exact same generic construction, as that would be inefficient, but the idea is the same). First, two new variables $x_\vee$ and $y_\vee$ are introduced together with an equation that makes sure that at least one of these variable is not equal to 1 ($e(g_0, x_\vee)e(g_0, y_\vee) = e(g_0, g_1)$). Then system 18 is replaced with a new system that is equivalent with 18 when $x_\vee \neq 1$ and is trivially satisfiable when $x_\vee = 1$. This is done by adding equation $e(\mathsf{ct}_1, x_\vee) = e(x_{\mathsf{ct}_1}, x_\vee)$ and replacing ct_1 with x_{ct_1} in 18. Next system 19 is replaced with a new system that is equivalent with 19 when $y_\vee \neq 1$ and is trivially satisfiable when $y_\vee = 1$. This is done by adding equation $e(g_0, y_\vee) = e(y_{g_0}, y_\vee)$ and replacing $[g_0\ x_4 x_5]$ with $[y_{g_0}\ x_4\ x_5]$. It can be observed that the resulting system of equations has

the desired properties:

$$
\begin{aligned}
e(g_0, x_\vee)e(g_0, y_\vee)e(g_0, g_1)^{-1} &= 1 \\
e(\mathsf{ct}_1, x_\vee)e(x_{\mathsf{ct}_1}, x_\vee)^{-1} &= 1 \\
e(x_{\mathsf{ct}_1}, g_1)e(x_1, g_1)^{-1} &= 1 \\
e(\mathsf{ct}_2, g_1)^{-1}e(x_2, g_1)e(x_3, g_1) &= 1 \\
x_1^{-1}g_0^{\phi} &= 1 \\
x_2^{-1}h_0^{\phi} &= 1 \\
e(g_0, y_\vee)e(y_{g_0}, y_\vee)^{-1} &= 1 \\
e(\mathsf{ct}_1, g_1)^{-1}e(x_6, g_1)e(x_4, g_1) &= 1 \\
e(\mathsf{ct}_2, g_1)^{-1}e(x_7, g_1)e(x_5, g_1) &= 1 \\
x_6^{-1}g_0^{\theta} &= 1 \\
x_7^{-1}h_0^{\theta} &= 1 \\
e(\mathbf{x}_8, \mathbf{vk}_4)^{-1}e([y_{g_0}\ x_4\ x_5], \mathbf{vk}_1)e(\mathbf{x}_9, \mathbf{vk}_2)e(\mathbf{x}_{10}, \mathbf{vk}_3) &= 1 \\
e(\mathbf{x}_9, x_{11})e(\mathbf{x}_{10}, g_1)^{-1} &= 1
\end{aligned}
\tag{20}
$$

The above set of equations, give rise to a Groth-Sahai NIWI proof of knowledge. It is easy to see that $\mathcal{T}_{\mathsf{Elgamal},\mathsf{KPW}}$ is domain transforming with respect to the above set of equations. Therefore we can make this NIWI proof of knowledge malleable, with the method described in Section 5.

G Comparison of our proof mauling method with the previous methods

In this section we compare our method for mauling Groth-Sahai proofs with that of [20] by applying both of them to a simple yet useful example. As shown in [20], this example arises when one tries to make a CM-CCA encryption scheme out of BBS encryption scheme by adding a Groth-Sahai proof to the ciphertext. Recall that in BBS encryption scheme, groups $\mathbb{G}$ and $\mathbb{G}_T$ of prime order p, generator g for $\mathbb{G}$ and a symmetric pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ are given; Private key consists of uniform random elements α and β from $\mathbb{Z}_p$ and public key is $(f = g^{\alpha}, h = g^{\beta})$. To encrypt a message $m \in \mathbb{G}$, one chooses uniform random elements r and s from $\mathbb{Z}_p$ and outputs $(u = f^r, v = h^s, w = g^{r+s}m)$.

The following system of pairing-product equations could be solved by one who has the plaintext and the encryption randomness.

$$
\begin{aligned}
&\mathsf{eq}_0 : e(u, g)e(x_1, f^{-1}) = 1 \\
&\mathsf{eq}_1 : e(v, g)e(x_2, h^{-1}) = 1 \\
&\mathsf{eq}_2 : e(w, g)e(x_3, g^{-1})e(x_1, g^{-1})e(x_2, g^{-1}) = 1
\end{aligned}
\tag{21}
$$

Consider the following mauling operation on the ciphertext: $(f^{\tilde{r}}u, h^{\tilde{s}}v, g^{\tilde{r}+\tilde{s}}w)$. We could consider this operation, as a mauling transformation acting on system of equations (21), giving the following system of equations:

$$\begin{aligned}
&\mathsf{eq}'_0 : e(f^{\tilde{r}}u, g)e(x'_1, f^{-1}) = 1 \\
&\mathsf{eq}'_1 : e(h^{\tilde{s}}v, g)e(x'_2, h^{-1}) = 1 \\
&\mathsf{eq}'_2 : e(g^{\tilde{r}+\tilde{s}}w, g)e(x_3, g^{-1})e(x'_1, g^{-1})e(x'_2, g^{-1}) = 1
\end{aligned} \tag{22}$$

[20] introduces 6 types of basic transformations that can be used to maul a Groth-Sahai proof. These types are:

1. Merging (multiplying) equations
2. Merging variables (adding a variable that is equal to the multiplication of a number of previous variables)
3. Exponentiating a variable
4. Adding a constant equation
5. Removing an equation
6. Removing a variable

We will not define these transformations here. Refer to [20] for the definitions. But this section can be followed without knowing the exact definitions and the meaning of the above transformations will become [almost] clear in the following discussion.

Let's denote the above transformation with T. In order to use [20]'s mauling method, we first need to decompose T as a composition of the above 6 mauling transformations. We first add three constant equations and get the following equations:

$$\begin{aligned}
&\mathsf{eq}_3 : e(f^{\tilde{r}}, g)e(g^{\tilde{r}}, f^{-1}) = 1 \\
&\mathsf{eq}_4 : e(h^{\tilde{s}}, g)e(g^{\tilde{s}}, h^{-1}) = 1 \\
&\mathsf{eq}_5 : e(g^{\tilde{r}+\tilde{s}}, g)e(g^{\tilde{r}}, g^{-1})e(g^{\tilde{s}}, g^{-1}) = 1
\end{aligned}$$

Next we merge eq_0 with eq_3, eq_1 with eq_4 and eq_2 with eq_5 to get the following of equations

$$\begin{aligned}
&\mathsf{eq}_6 : e(u, g)e(x_1, f^{-1})e(f^{\tilde{r}}, g)e(g^{\tilde{r}}, f^{-1}) = 1 \\
&\mathsf{eq}_7 : e(v, g)e(x_2, h^{-1})e(h^{\tilde{s}}, g)e(g^{\tilde{s}}, h^{-1}) = 1 \\
&\mathsf{eq}_8 : e(w, g)e(x_3, g^{-1})e(x_1, g^{-1})e(x_2, g^{-1})e(g^{\tilde{r}+\tilde{s}}, g)e(g^{\tilde{r}}, g^{-1})e(g^{\tilde{s}}, g^{-1}) = 1
\end{aligned}$$

Next we use the second type of transformation and merge constant $g^{\tilde{r}}$ with variable x_1 into the new variable x'_1 and merge constant $g^{\tilde{s}}$ with variable x_2 into the new variable x'_2. This merge operations result in the following equations (equations eq_9 and eq_{10} correspond to the first merge and the other two correspond to the second merge).

$$\mathsf{eq}_9 : e(x_1, f)e(g^{\tilde{r}}, f)e(x'_1, f^{-1}) = 1$$

$$\mathsf{eq}_{10} : e(x_1, g)e(g^{\tilde{r}}, g)e(x'_1, g^{-1}) = 1$$
$$\mathsf{eq}_{11} : e(x_2, h)e(g^{\tilde{s}}, h)e(x'_2, h^{-1}) = 1$$
$$\mathsf{eq}_{12} : e(x_2, g)e(g^{\tilde{s}}, g)e(x'_2, g^{-1}) = 1$$

Next we merge equation eq_6 with equation eq_9, equation eq_7 with equation eq_{11} and equations eq_{10} and eq_{12} with equation eq_8 and reach at the following equations:

$$\mathsf{eq}_{13} : e(x'_1, f^{-1})e(f^{\tilde{r}}u, g) = 1$$
$$\mathsf{eq}_{14} : e(x'_2, h^{-1})e(h^{\tilde{s}}v, g) = 1$$
$$\mathsf{eq}_{15} : e(g^{\tilde{r}+\tilde{s}}w, g)e(x_3, g^{-1})e(x'_1, g^{-1})e(x'_2, g^{-1}) = 1$$

Next we use the equation-remove and variable-remove transformations to remove all of the unnecessary equations and variables and just keep equations eq_{13}, eq_{14} and eq_{15} which make the desired mauled system of equations. Now that we have a sequence of transformations that take the initial system of equations to the final system of equations, we can maul a proof for the satisfiability of the initial system into a proof for the satisfiability of the final system, by applying the proof mauling algorithms corresponding to the above sequence of transformations. This is the method used in [20] to make a CM-CCA encryption scheme out of BBS encryption scheme by adding a malleable Groth-Sahai proof.

With our proof mauling method, this can be done in a single step. We have to show that T is domain-transforming. Informally, this means that the change made to the constants by T can be cancelled by an efficient change to the variables, in the following sense[13]:

$$e(u, g)e(x_1, f^{-1}) = e(f^{\tilde{r}}u, g)e(x'_1, f^{-1})$$
$$e(v, g)e(x_2, h^{-1}) = e(h^{\tilde{s}}v, g)e(x'_2, h^{-1})$$
$$e(w, g)e(x_3, g^{-1})e(x_1, g^{-1})e(x_2, g^{-1}) = e(g^{\tilde{r}+\tilde{s}}w, g)e(x_3, g^{-1})e(x'_1, g^{-1})e(x'_2, g^{-1})$$

This can be achieved simply by taking $x'_1 = g^{\tilde{r}}x_1$ and $x'_2 = g^{\tilde{s}}x_2$. Having this, the mauling of the proof is also simple. Let $(C_1, C_2, C_3, \pi_1, \pi_2, \psi_1, \psi_2)$ be a proof for the satisfiability of the original system of equations[14]. The mauled proof would be $((1, g^{\tilde{r}})C_1, (1, g^{\tilde{s}})C_2, C_3, \pi_1, \pi_2, \psi_1, \psi_2)$.

H A method for mauling Groth-Sahai: General case

In Section 5 we review Groth-Sahai proofs and introduce our proof mauling method only for pairing-product equations. In this section we show that our

[13] More accurately, we have to show this equivalence for the embedding of the equations (as defined in Section 5) but in this case, the embedding version of the equivalence immediately follows.

[14] Refer to Section 5 or [29], to see why the proof has this format.

method is also applicable to other types of equations. To express this concisely, we use [27]'s unified notation for representing different types of equations. Hence, unlike rest of the paper, in this section we use additive notation for group operation.

Groth-Sahai proofs: a review. First, we review the notation that [27]'s authors use to give a unified treatment of pairing-product, multi-scalar production, and quadratic equations. In their notation, it is assumed that $\mathbb{Z}_p$-vector-spaces $\mathbb{A}_1$, $\mathbb{A}_2$, $\mathbb{A}_T$, $\mathbb{B}_1$, $\mathbb{B}_2$ and $\mathbb{B}_T$ are given together with efficient bilinear maps $f : \mathbb{A}_1 \times \mathbb{A}_2 \to \mathbb{A}_T$ and $E : \mathbb{B}_1 \times \mathbb{B}_2 \to \mathbb{B}_T$. It is also assumed that there are efficient inclusion maps $\iota_i : \mathbb{A}_i \to \mathbb{B}_i$ for $i \in \{1, 2, T\}$, such that:

$$E(\iota_1(a_1), \iota_2(a_2)) = \iota_T(f(a_1, a_2)).$$

The inclusion map ι_i is extended to $\mathbb{A}_i^n$ in the trivial way. Bilinear maps f and E are extended to $\mathbb{A}_1^n \times \mathbb{A}_2^n$ and $\mathbb{B}_1^n \times \mathbb{B}_2^n$, respectively, as follows:

$$f((a_i)_{i=1}^n, (b_i)_{i=1}^n) = \sum_{i=1}^{n} f(a_i, b_i) \quad , \quad E((x_i)_{i=1}^n, (y_i)_{i=1}^n) = \sum_{i=1}^{n} E(x_i, y_i)$$

We are only interested in SXDH-based proofs, in which case $\mathbb{B}_1 = \mathbb{G}_1^2$, $\mathbb{B}_2 = \mathbb{G}_2^2$ and $\mathbb{B}_T = \mathbb{G}_T^4$ and

$$E((x_1, x_2), (y_1, y_2)) = (e(x_1, y_1), e(x_1, y_2), e(x_2, y_1), e(x_2, y_2)),$$

where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order p and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear pairing. Unlike $\mathbb{B}_1, \mathbb{B}_1$ and $\mathbb{B}_T$, as we will explain below, the instantiation of $\mathbb{A}_1$, $\mathbb{A}_2$, $\mathbb{A}_T$, and also ι_1, ι_2 and ι_T will depend on the type of equation (pairing-product, multi-scalar multiplication or quadratic) and because we will use all of these equation types in our use case, we will not reduce the generality anymore.

The proof system's crs can be generated in two modes. In one mode the proof has perfect soundness and in the other mode it has perfect witness-indistinguishability. In the SXDH-based proofs that we use in this paper, these two modes are as follows:

- **Perfect soundness:** In this mode crs consists of $(u_1, u_2, u, v_1, v_2, v)$ where u and u_i are members of $\mathbb{G}_1^2$ and v and v_i are members of $\mathbb{G}_2^2$ such that $\langle u_1, u_2 \rangle$ (span of u_1 and u_2) and $\langle u, u_1 \rangle$ are one and two-dimensional subspaces of $\mathbb{G}_1^2$, respectively, and $\langle v_1, v_2 \rangle$ and $\langle v, v_1 \rangle$ are one and two-dimensional subspaces of $\mathbb{G}_2^2$, respectively.
- **Perfect witness-indistinguishability:** This mode is similar to above, except that $\langle u, u_1 \rangle$ and $\langle v, v_1 \rangle$ are one-dimensional whereas $\langle u_1, u_2 \rangle$ and $\langle v_1, v_2 \rangle$ are two-dimensional.

The indistinguishability of these two modes, follows from the DDH assumption in $\mathbb{G}_1$ and $\mathbb{G}_2$. Mappings f, ι_i and vector spaces $\mathbb{A}_i$ will be as follows:

– In case of **pairing-product** equations, $f(a,b) = e(a,b)$, $\mathbb{A}_i = \mathbb{G}_i$, $\iota_1(a) = (0,a)$, $\iota_2(b) = (0,b)$ and $\iota_T(c) = (0,0,0,c)$.
– In case of **multi-scalar multiplication in** $\mathbb{G}_1$, $f(a,b) = [b]a$, $\mathbb{A}_1 = \mathbb{G}_1$, $\mathbb{A}_2 = \mathbb{Z}_p$, $\mathbb{A}_T = \mathbb{G}_1$, $\iota_1(a) = (0,a)$, $\iota_2(b) = [b]v$ and $\iota_T(c) = E((0,c),v)$.
– In case of **multi-scalar multiplication in** $\mathbb{G}_2$, $f(a,b) = [a]b$, $\mathbb{A}_1 = \mathbb{Z}_p$, $\mathbb{A}_2 = \mathbb{G}_2$, $\mathbb{A}_T = \mathbb{G}_2$, $\iota_1(a) = [a]u$, $\iota_2(b) = (0,b)$ and $\iota_T(c) = E(u,(0,c))$.
– In case of **quadratic** equations, $f(a,b) = ab$, $\mathbb{A}_1 = \mathbb{Z}_p$, $\mathbb{A}_2 = \mathbb{Z}_p$, $\mathbb{A}_T = \mathbb{Z}_p$, $\iota_1(a) = [a]u$, $\iota_2(b) = [b]v$ and $\iota_T(c) = [c]E(u,v)$.

We are interested in a system of equations of the following form:

$$f(\mathbf{x}, \boldsymbol{\Gamma}_l \mathbf{y}) + f(\mathbf{a}_l, \mathbf{y}) + f(\mathbf{x}, \mathbf{b}_l) = t_l, \quad 1 \leqslant l \leqslant L \tag{23}$$

where $\mathbf{x} \in \mathbb{A}_1^m$ and $\mathbf{y} \in \mathbb{A}_2^n$ are variables and $\mathbf{a}_l \in \mathbb{A}_1^n$, $\mathbf{b}_l \in \mathbb{A}_2^m$ and $\boldsymbol{\Gamma}_l \in \mathbb{Z}_p^{m \times n}$ are constants.

To define these elements, $\mathbf{U}$ and $\mathbf{V}$, defined below, are needed.

$$\mathbf{U} = \begin{cases} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \in \mathbb{B}_1^2 & \text{if } \mathbb{A}_1 = \mathbb{G}_1 \\ \\ \begin{bmatrix} u_1 \end{bmatrix} \in \mathbb{B}_1 & \text{if } \mathbb{A}_1 = \mathbb{Z}_p \end{cases}, \quad \mathbf{V} = \begin{cases} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \in \mathbb{B}_2^2 & \text{if } \mathbb{A}_2 = \mathbb{G}_2 \\ \\ \begin{bmatrix} v_1 \end{bmatrix} \in \mathbb{B}_2 & \text{if } \mathbb{A}_2 = \mathbb{Z}_p \end{cases}$$

Let $\hat{m}$ and $\hat{n}$ be $\mathbf{U}$ and $\mathbf{V}$'s row count, respectively.

A Groth-Sahai proof for the satisfiability of 23 consists of a set of commitments and a set of proof elements. Commitments are defined as follows:

$$\mathbf{C} = \iota_1(\mathbf{x}) + \mathbf{R}\mathbf{U} \quad , \quad \mathbf{D} = \iota_2(\mathbf{y}) + \mathbf{S}\mathbf{V}, \tag{24}$$

where $\mathbf{R} \leftarrow \mathbb{Z}_p^{m \times \hat{m}}$ and $\mathbf{S} \leftarrow \mathbb{Z}_p^{n \times \hat{n}}$. Proof elements are defined as follows:

$$\boldsymbol{\Pi}_l = \mathbf{R}^T \boldsymbol{\Gamma}_l \mathbf{D} + \mathbf{R}^T \iota_2(\mathbf{b}_l) + \mathbf{T}_l^T \mathbf{V} \quad , \quad \boldsymbol{\Psi}_l = \mathbf{S}^T \boldsymbol{\Gamma}_l^T \iota_1(\mathbf{x}) + \mathbf{S}^T \iota_1(\mathbf{a}_l) - \mathbf{T}_l \mathbf{U}, \tag{25}$$

where $\mathbf{T}_l \leftarrow \mathbb{Z}_p^{\hat{n} \times \hat{m}}$, except when $\boldsymbol{\Gamma}_l = \mathbf{0} \wedge (\mathbf{a}_l = \mathbf{0} \vee \mathbf{b}_l = \mathbf{0})$, in which case $\mathbf{T}_l = \mathbf{0}$. The verifier checks the following:

$$E(\mathbf{C}, \boldsymbol{\Gamma}_l \mathbf{D}) + E(\iota_1(\mathbf{a}_l), \mathbf{D}) + E(\mathbf{C}, \iota_2(\mathbf{b}_l)) - \iota_T(t_l) = E(\mathbf{U}, \boldsymbol{\Pi}_l) + E(\boldsymbol{\Psi}_l, \mathbf{V}), 1 \leqslant l \leqslant L \tag{26}$$

The above proof system is witness-indistinguishable. 23 is called tractable if there is an efficient algorithm that finds a solution to 23 in which a variable over $\mathbb{Z}_p$ may have different values in different equations but a variable over $\mathbb{G}_i$ has the same value in all equations. If 23 is tractable, then the proof system given above is zero-knowledge.

Our mauling method. To define the class of transformations that we consider in mauling proofs, we set a few more notations.

Notation 3 *If* $\mathsf{coef} = \big((\boldsymbol{\Gamma}_l, \mathbf{a}_l, \mathbf{b}_l, t_l)_{l=1\cdots L}, m, n\big)$*, then*

$$\begin{cases} F_{\mathsf{coef}} : \mathbb{B}_1^m \times \mathbb{B}_2^n \to \mathbb{B}_T \\ F_{\mathsf{coef}}(\mathbf{X}, \mathbf{Y}) = \Big(E(\mathbf{X}, \boldsymbol{\Gamma}_l \mathbf{Y}) + E(\iota_1(\mathbf{a}_l), \mathbf{Y}) + E(\mathbf{X}, \iota_2(\mathbf{b}_l)) - \iota_T(t_l)\Big)_{l=1\cdots L} \end{cases}$$

–

$$\begin{cases} f_{\mathsf{coef}} : \mathbb{A}_1^m \times \mathbb{A}_2^n \to \mathbb{A}_T \\ f_{\mathsf{coef}}(\mathbf{x}, \mathbf{y}) = \Big(f(\mathbf{x}, \boldsymbol{\Gamma}_l \mathbf{y}) + f(\mathbf{a}_l, \mathbf{y}) + f(\mathbf{x}, \mathbf{b}_l) - t_l\Big)_{l=1\cdots L} \end{cases}$$

– *We call R a Groth-Sahai relation if it contains tuples $\Big(\mathsf{coef}, (\mathbf{x}, \mathbf{y})\Big)$ where* coef *is of the previously described form and $f_{\mathsf{coef}}(\mathbf{x}, \mathbf{y}) = 0$.*

Definition 4. *Let R be Groth-Sahai relation and $T = (T_1, T_2)$ be a transformation over R. We call T domain-transforming if there is an efficient function $I_T = (I_{T,x}, I_{T,y})$ such that for every* $\mathsf{coef} \in L$*, if* $\mathsf{coef}' = T_1(\mathsf{coef})$*, then $F_{\mathsf{coef}'}(I_{T,x}(\mathbf{X}), I_{T,y}(\mathbf{Y})) \equiv F_{\mathsf{coef}}(\mathbf{X}, \mathbf{Y})$.*

Below, we give a method for mauling Groth-Sahai proofs by domain-transforming transformations.

Construction 8 *Let R be a Groth-Sahai relation and $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ be the Groth-Sahai NIWI proof system for R. Let $\mathcal{T}$ be a set of domain-transforming transformations over R. By adding the following algorithm, we make the proof system malleable with respect to $\mathcal{T}$.*

– $\mathsf{ZKEval}(\mathsf{crs}, T, \pi)$*: Let $I_T = (I_{T,x}, I_{T,y})$ be the efficient algorithm that 4 specifies. Parse $\pi = (\mathbf{C}, \mathbf{D}, (\boldsymbol{\Pi}_l, \boldsymbol{\Psi}_l)_{l=1\cdots L})$, set $\widetilde{\mathbf{C}} = I_{T,x}(\mathbf{C})$, $\widetilde{\mathbf{D}} = I_{T,y}(\mathbf{D})$, $\widetilde{\boldsymbol{\Pi}}_l = \boldsymbol{\Pi}_l$, $\widetilde{\boldsymbol{\Psi}}_l = \boldsymbol{\Psi}_l$ and output $\widetilde{\pi} = (\widetilde{\mathbf{C}}, \widetilde{\mathbf{D}}, (\widetilde{\boldsymbol{\Pi}}_l, \widetilde{\boldsymbol{\Psi}}_l)_{l=1\cdots L})$.*

Theorem 9. *Construction 8 is a NIWI for R that is malleable with respect to $\mathcal{T}$.*

Proof. Let $\mathsf{coef} = ((\boldsymbol{\Gamma}_l, \mathbf{a}_l, \mathbf{b}_l, t_l)_{l=1\cdots L}, m, n)$ be a valid problem instance and $\pi = (\mathbf{C}, \mathbf{D}, (\boldsymbol{\Pi}_l, \boldsymbol{\Psi}_l)_{l=1\cdots L})$ be valid proof. Hence

$$E(\mathbf{C}, \boldsymbol{\Gamma}_l \mathbf{D}) + E(\iota_1(\mathbf{a}_l), \mathbf{D}) + E(\mathbf{C}, \iota_2(\mathbf{b}_l)) - \iota_T(t_l) = E(\mathbf{U}, \boldsymbol{\Pi}_l) + E(\boldsymbol{\Psi}_l, \mathbf{V}), \quad , \quad 1 \leqslant l \leqslant L$$

which can be rewritten as $F_{\mathsf{coef}}(\mathbf{C}, \mathbf{D}) = (E(\mathbf{U}, \boldsymbol{\Pi}_l) + E(\boldsymbol{\Psi}_l, \mathbf{V}))_{l=1\cdots L}$. Based on T being domain-transforming, if $\mathsf{coef}' = T_1(\mathsf{coef})$ then we have $F_{\mathsf{coef}'}(I_{T,x}(\mathbf{C}), I_{T,y}(\mathbf{D})) = (E(\mathbf{U}, \boldsymbol{\Pi}_l) + E(\boldsymbol{\Psi}_l, \mathbf{V}))_{l=1\cdots L}$ which means that $\widetilde{\pi}$ verifies as a proof for coef'.