Universally Composable Two-Server PAKE

Franziskus Kiefer[1] and Mark Manulis[2]

[1] Mozilla
Berlin, Germany
mail@franziskuskiefer.de
[2] Surrey Center for Cyber Security
Department of Computer Science, University of Surrey, UK
mark@manulis.eu

Abstract. Two-Server Password Authenticated Key Exchange (2PAKE) protocols apply secret sharing techniques to achieve protection against server-compromise attacks. 2PAKE protocols eliminate the need for password hashing and remain secure as long as one of the servers remains honest. This concept has also been explored in connection with two-server password authenticated secret sharing (2PASS) protocols for which game-based and universally composable versions have been proposed. In contrast, universally composable PAKE protocols exist currently only in the single-server scenario and all proposed 2PAKE protocols use game-based security definitions.
In this paper we propose the first construction of an universally composable 2PAKE protocol, alongside with its ideal functionality. The protocol is proven UC-secure in the standard model, assuming a common reference string which is a common assumption to many UC-secure PAKE and PASS protocols. The proposed protocol remains secure for arbitrary password distributions. As one of the building blocks we define and construct a new cryptographic primitive, called Trapdoor Distributed Smooth Projective Hash Function (TD-SPHF), which could be of independent interest.

1 Introduction

Password Authenticated Key Exchange (PAKE) protocols have been extensively researched over the last twenty years. They allow two protocol participants sharing a low-entropy secret (password) to negotiate an authenticated secret key. Several PAKE security models are widely used such as the game-based PAKE model, called BPR, by Bellare, Pointcheval and Rogaway [8,4] and the PAKE model in the Universal Composability (UC) framework by Canetti [19]. PAKE protocols are often considered in a client-server scenario where the client password is registered and stored in a protected way on the server side such that it can be used later to authenticate the client. This approach however leads to an intrinsic weakness of single-server PAKE protocols against server-compromise attacks. An attacker who breaks into the server can efficiently recover client's password and impersonate the client to the server as well as to other servers if this password is used across many client accounts which is often the case. A number of approaches have been proposed to alleviate this threat. For instance, verifier-based PAKE [24,38,12], also known as augmented PAKE [9], considers an asymmetric setting in which the server uses a randomized password hash to verify a client holding the corresponding password. The crucial weakness of VPAKE protocols is that they do not protect against offline dictionary attacks on compromised password hashes, i.e. an attacker can still recover the password, which can often be done efficiently with current tools like [25,34].

Two-server PAKE (2PAKE) protocols solve this problem through secret sharing techniques. The client password is split into two shares and each server receives its own share upon registration. In order to authenticate the client both servers take part in the protocol execution. 2PAKE security typically holds against an active attacker who can compromise at most one server and thus learn the corresponding password share. 2PAKE protocols can be symmetric (e.g. [13,37,29,31]) where both servers compute the same session key and asymmetric (e.g. [29]) where each server can compute an independent session key with the client or assist another server in the authentication process [39,28] without computing the key. A potential drawback of symmetric protocols is that by corrupting one server the attacker may use learned key material to read

communications between the client and the other server. Existing 2PAKE protocols were analysed using variants of the BPR model and do not offer compositional security guarantees. While 2PAKE can be seen as a special case of Threshold PAKE (TPAKE), e.g. [36,32], that adopt t-out-of-n secret sharing, existing TPAKE protocols do not necessarily provide solutions for 2PAKE, e.g. [36] requires $t < n/3$. Finally, we note that UC-security was considered for a class of Two-Server/Threshold Password Authenticated Secret Sharing (2/TPASS) protocols, e.g. [15,26,14], that address a different problem of sharing a chosen key across multiple servers and its subsequent reconstruction from the password.

In this paper we propose the first UC-secure (asymmetric) 2PAKE protocol where one of the two servers computes an independent session key with the client. We rely on a common reference string, which is a standard assumption for UC-secure PAKE protocols. As a consequence of UC modeling our protocol offers security for all password distributions, which is notoriously difficult to achieve in BPR-like models. One challenge in achieving UC security is that the protocol must remain simulatable against active attackers that play with a correctly guessed password (unlike in game-based models where simulation can be aborted). In order to achieve simulatability we introduce a new building block, called *Trapdoor Distributed Smooth Projective Hash Functions (TD-SPHF)*, offering distributed SPHF properties from [31] and the SPHF trapdoor property from [10]. While traditional SPHF were used in the design of single-server PAKE protocols, the 2PAKE protocol framework from [31], a generalisation of [29] that was proven secure in the BPR-like model, required an extension of SPHF to a distributed setting. Such distributed SPHF alone are not sufficient for achieving the UC security. Our TD-SPHF helps to achieve simulatability for 2PAKE protocols and could be of independent interest for other UC-secure constructions.

2 Preliminaries and Building Blocks

Our 2PAKE protocol is defined over bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order q with an efficiently computable map $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$. The following properties have to hold: i) If g_1 is a generator of $\mathbb{G}_1$ and g_2 is a generator of $\mathbb{G}_2$, then $e(g_1, g_2)$ is a generator of $\mathbb{G}_T$. ii) For generators g_1, g_2 and scalar $x \in_R \mathbb{Z}_q$ it holds that $e(g_1^x, g_2) = e(g_1, g_2^x) = e(g_1, g_2)^x$. We require further that the Symmetric External Diffie-Hellman assumption (SXDH) ([6,5] amongst others) holds in those groups. SXDH states that the DDH problem is hard in $\mathbb{G}_1$ and $\mathbb{G}_2$. It is believed that the SXDH assumption holds in MNT curves (named after Miyaji, Nakabayashi, and Takano [33], denote prime-order curves with embedding degree 3, 4 or 6), and pairings could be implemented with Tate pairings. All computations defined on a q-order group in the following are performed in $\mathbb{G}_1$. Let λ denote the security parameter throughout this work.

Commitments Let $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com})$ denote an efficient commitment scheme and $C \leftarrow \mathtt{Com}(x; r)$ a commitment on x using randomness r, with $\mathtt{CSetup}$ generating parameters for $\mathtt{C}$.[3] A commitment scheme $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com})$ is *efficient* if $\mathtt{CSetup}(\lambda)$ and $(C, d) \leftarrow \mathtt{Com}(x; r)$ are computable in polynomial time, *complete* if $\mathtt{Com}(d) = (C, d)$ for $(C, d) \leftarrow \mathtt{Com}(x; r)$, and secure if it is

- Binding: For all PPT adversaries $\mathcal{A}$ there exists a negligible function $\varepsilon_{\mathrm{bi}}(\cdot)$ such that for all $(x, x', r, r', C) \leftarrow \mathcal{A}$: $\Pr[x \neq x' \wedge (C, d) = \mathtt{Com}(x; r) \wedge (C, d') = \mathtt{Com}(x'; r')] \leq \varepsilon_{\mathrm{bi}}(\lambda)$,
- Hiding: For all PPT adversaries $\mathcal{A}$ there exists a negligible function $\varepsilon_{\mathrm{hi}}(\cdot)$ such that for all x_0, x_1 with $|x_0| = |x_1|$ and $b \in_R \{0, 1\}$, $(C, d) \leftarrow \mathtt{Com}(x_b; r)$ and $b' \leftarrow \mathcal{A}(C, x_1, x_2)$: $\Pr[b = b'] \leq 1/2 + \varepsilon_{\mathrm{hi}}(\lambda)$.

Instantiation [35] We will use perfectly hiding, computationally binding, homomorphic Pedersen commitments [35] defined as follows. Let $\mathtt{C_P} = (\mathtt{CSetup}, \mathtt{Com})$ with $(g, h, q, \lambda) \leftarrow \mathtt{CSetup}(\lambda)$ and $C \leftarrow \mathtt{Com} = (x; r) = g^x h^r$ denote the Pedersen commitment scheme where g and h are generators of a cyclic group $\mathbb{G}$ of prime-order q with bit-length in the security parameter λ and the discrete logarithm of h with respect to base g is not known. Pedersen commitments are *additively homomorph*, i.e. for all $(C_i, d_i) \leftarrow \mathtt{Com}(x_i; r_i)$ for $i \in 0, \ldots, m$ it holds that $\prod_{i=0}^{m} C_i = \mathtt{Com}(\sum_{i=0}^{m} x_i; \sum_{i=0}^{m} r_i)$.

[3] We usually omit decommitment d ouptut by $\mathtt{Com}(x; r)$.

Committed Zero-Knowledge Proofs We use committed Σ-protocols for security against malicious verifiers [22,27]. Note that we do not require extractability (proof of knowledge) here, which allows us to avoid the necessity of rewinding. A zero-knowledge proof ZKP is executed between a prover and a verifier, proving that a word x is in a language L, using a witness w proving so.[4] An interactive protocol ZKP for a language L between prover P and verifier V is a zero knowledge proof if the following holds:

- Completeness: If $x \in L$, V accepts if P holds a witness proving so.
- Soundness: For every malicious prover $P^*(x)$ with $x \in L$ the probability of making V accept is negligible.
- Zero-Knowledge: If $x \in L$, then there exists an efficient simulator that on input of x is able to generate a view, indistinguishable from the view of a malicious verifier V^*.

Let $P_1(x, w, r)$ and $P_2(x, w, r, c)$ denote the two prover steps of a Σ-protocol and $H : \{0,1\}^* \mapsto \mathbb{Z}_q$ a collision-resistant hash function. A committed Σ-protocol is then given by the following four steps:

- The prover computes the first message $\mathtt{Co} \leftarrow P_1(x, w, r)$ and $m_1 \leftarrow \mathtt{Com}(H(x, \mathtt{Co}); r_1) = g^{H(x,\mathtt{Co})} h^{r_1}$, and sends m_1 to the verifier.
- The verifier chooses challenge $\mathtt{Ch} = c \in_R \mathbb{Z}_q$ and returns it to the prover.
- The prover computes the second message $\mathtt{Rs} \leftarrow P_2(x, w, r, c)$ and $m_2 \leftarrow \mathtt{Com}(H(\mathtt{Rs}); r_2) = g^{H(\mathtt{Rs})} h^{r_2}$, and sends m_2 to the verifier.
- Further, the prover opens the commitments m_1 and m_2 sending $(x, \mathtt{Co}, \mathtt{Rs}, r_1, r_2)$ to the verifier.
- The verifier accepts iff both commitments are valid and if the verification of the Σ-protocol $(x, \mathtt{Com}, \mathtt{Ch}, \mathtt{Rs})$ is successful.

Cramer-Shoup Encryption with Labels Let $C = (\ell, \boldsymbol{u}, e, v) \leftarrow \mathtt{Enc}^{\mathrm{CS}}_{\mathtt{pk}}(\ell, m; r)$ (on label ℓ, message m, and randomness r) with $\boldsymbol{u} = (u_1, u_2) = (g_1^r, g_2^r)$, $e = h^r g_1^m$ and $v = (cd^{\xi})^r$ with $\xi = H_k(\ell, \boldsymbol{u}, e)$ denote a labelled Cramer-Shoup ciphertext. We assume $m \in \mathbb{Z}_q$ and $\mathbb{G}$ is a cyclic group of prime order q with generators g_1 and g_2 such that $g_1^m \in \mathbb{G}$. The CS public key is defined as $\mathtt{pk} = (p, \mathbb{G}, g_1, g_2, c, d, H_k)$ with $c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2}, h = g_1^z$ and hash function H_k such that $\tau = (x_1, x_2, y_1, y_2, z)$ denotes the decryption key. Decryption is defined as $g_1^m = \mathtt{Dec}^{\mathrm{CS}}_{\mathtt{dk}}(C) = e/u_1^z$ if $u_1^{x_1+y_1\cdot\xi'} u_2^{x_2+y_2\cdot\xi'} = v$ with $\xi' = H_k(\ell, \boldsymbol{u}, e)$.

2.1 Smooth Projective Hashing (SPHF)

First, we recall definitions for classical SPHF tailored to the PAKE use-case and cyclic groups $\mathbb{G}$ of prime-order q. We use languages of ciphertexts with the password as message and the randomness as witness. An SPHF language L for a given password pw from dictionary $\mathcal{D}$ is given by L_{pw}. The public parameter of the language is the common reference string $\mathtt{crs}$ containing the public key $\mathtt{pk}$ of the encryption scheme. By τ we denote the $\mathtt{crs}$ trapdoor, the secret key to $\mathtt{pk}$. Let $\mathcal{L}$ be the encryption scheme used to generate words in L_{pw}. Unless stated otherwise we assume that $\mathcal{L}$ is a labelled CCA-secure encryption scheme, e.g. labelled Cramer-Shoup scheme.

Definition 1 (Languages of Ciphertexts). *Let $L_{\mathrm{pw}} \subseteq \{(\ell, C, \mathrm{pw}^*)\} = \mathcal{C}$ denote the language of labelled ciphertexts under consideration with ciphertext (ℓ, C) under $\mathtt{pk}$ and password $\mathrm{pw}^* \in \mathcal{D}$. A ciphertext C is in language L_{pw} iff there exists randomness r such that $C \leftarrow \mathtt{Enc}^{\mathcal{L}}_{\mathtt{pk}}(\ell, \mathrm{pw}; r)$.*

Smooth projective hashing for languages of ciphertexts where the projection key does not depend on the ciphertext is defined as follows (see also [30,10]).

Definition 2 (KV-SPHF). *Let L_{pw} denote a language of ciphertexts such that $C \in L_{\mathrm{pw}}$ if there exists randomness r proving so. A* smooth projective hash function *for ciphertext language L_{pw} consists of the following four algorithms:*

[4] Zero-knowledge languages L are independent from the smooth projective hashing languages introduced in Section 2.1.

- $\mathtt{KGen_H}(L_{\mathrm{pw}})$ *generates a random hashing key* $\mathtt{k_h}$ *for language* L_{pw}.
- $\mathtt{KGen_P}(\mathtt{k_h}, L_{\mathrm{pw}})$ *derives the projection key* $\mathtt{k_p}$ *from hashing key* $\mathtt{k_h}$.
- $\mathtt{Hash}(\mathtt{k_h}, L_{\mathrm{pw}}, C)$ *computes hash value* h *from hashing key* $\mathtt{k_h}$ *and ciphertext* C.
- $\mathtt{PHash}(\mathtt{k_p}, L_{\mathrm{pw}}, C, r)$ *computes hash value* h *from projection key* $\mathtt{k_p}$*, ciphertext* C *and randomness* r.

A SPHF has to fulfil the following three properties:

- *Correctness*: If $C \in L$, with r proving so, then $\mathtt{Hash}(\mathtt{k_h}, L_{\mathrm{pw}}, C) = \mathtt{PHash}(\mathtt{k_p}, L_{\mathrm{pw}}, C, r)$.
- *Smoothness*: If $\{(\ell, C, \mathrm{pw}^*)\} \ni C \notin L_{\mathrm{pw}}$, the hash value h is (statistically) indistinguishable from a random element.
- *Pseudorandomness*: If $C \in L_{\mathrm{pw}}$, the hash value h is (computationally) indistinguishable from a random element.

We refer to the original work or Appendix A for more details.

2.2 Trapdoor Smooth Projective Hashing

For efficient one-round UC-secure PAKE a new SPHF flavor, called Trapdoor SPHF (T-SPHF), was introduced in [10]. T-SPHF adds three additional functions to the classical SPHF definition allowing computation of the hash value from the projection key, ciphertext and trapdoor τ'.[5]

Definition 3 (Trapdoor SPHF). *Let* L_{pw} *denote a language of ciphertexts such that* $C \in L_{\mathrm{pw}}$ *if there exists randomness* r *proving so. A* trapdoor smooth projective hash function *for a ciphertext language* L_{pw} *consists of the following seven algorithms:*

- $\mathtt{KGen_H}$, $\mathtt{KGen_P}$, $\mathtt{Hash}$ *and* $\mathtt{PHash}$ *are as given in Definition 2*
- $\mathtt{TSetup}(\mathtt{crs})$ *generates a second* $\mathtt{crs}'$ *with trapdoor* τ' *on input of a* $\mathtt{crs}$
- $\mathtt{VerKp}(\mathtt{k_p}, L_{\mathrm{pw}})$ *returns* 1 *iff* $\mathtt{k_p}$ *is a valid projection key,* 0 *otherwise*
- $\mathtt{THash}(\mathtt{k_p}, L_{\mathrm{pw}}, C, \tau')$ *computes the hash value* h *of* C *using the projection key* $\mathtt{k_p}$ *and trapdoor* τ'

We assume $\mathtt{crs}'$ *is, like* $\mathtt{crs}$*, made available to all parties.*

For more details on T-SPHF see the original work or Appendix B.

2.3 Distributed Smooth Projective Hashing

Another flavor, called Distributed SPHF (D-SPHF), was introduced in [31] for use in (non-composable) 2PAKE protocols such as [29] where servers hold password shares pw_1 and pw_2 respectively, and the client holds $\mathrm{pw} = \mathrm{pw}_1 + \mathrm{pw}_2$. For a more general description see [31]. Due to the nature of the words considered in D-SPHF they produce two different hash values. One can think of the two hash values as h_0 for C_0 (from the client) and h_x for C_1, C_2 (from the two servers). The hash value h_0 can be either computed with knowledge of the client's hash key $\mathtt{k_{h0}}$ or with the server's witnesses r_1, r_2 that C_1, C_2 are in L_{pw_i}, $i \in \{1, 2\}$ respectively. The hash value h_x can be computed with knowledge of the server hash keys $\mathtt{k_{h1}}, \mathtt{k_{h2}}$ or with the client's witness r_0 that C_0 is in L_{pw}. The combined language is denoted by $L_{\widehat{\mathrm{pw}}}$.

Definition 4 (Distributed SPHF). *Let* $L_{\widehat{\mathrm{pw}}}$ *denote a language such that* $C = (C_0, C_1, C_2) \in L_{\widehat{\mathrm{pw}}}$ *if there exists a witness* $r = (r_0, r_1, r_2)$ *proving so,* $\mathrm{pw} = \mathrm{pw}_1 + \mathrm{pw}_2$ *and there exists a function* $\mathtt{Dec}'$ *such that* $\mathtt{Dec}'(C_1 C_2) = \mathtt{Dec}'(C_0)$*. A distributed smooth projective hash function for language* $L_{\widehat{\mathrm{pw}}}$ *consists of the following six algorithms:*

- $\mathtt{KGen_H}(L_{\widehat{\mathrm{pw}}})$ *generates a hashing key* $\mathtt{k_{h}}i$ *for* $i \in \{0, 1, 2\}$ *and language* $L_{\widehat{\mathrm{pw}}}$.
- $\mathtt{KGen_P}(\mathtt{k_{h}}i, L_{\widehat{\mathrm{pw}}})$ *derives projection key* $\mathtt{k_{p}}_i$ *from hashing key* $\mathtt{k_{h}}i$ *for* $i \in \{0, 1, 2\}$.

[5] Note that τ' is a different trapdoor than the CRS trapdoor τ.

- $\texttt{Hash}_x(\texttt{k}_{\texttt{h}0}, L_{\widehat{\text{pw}}}, C_1, C_2)$ *computes hash value* h_x *from hashing key* $\texttt{k}_{\texttt{h}0}$ *and two server ciphertexts* C_1 *and* C_2*.*
- $\texttt{PHash}_x(\texttt{k}_{\texttt{p}0}, L_{\widehat{\text{pw}}}, C_1, C_2, r_1, r_2)$ *computes hash value* h_x *from projection key* $\texttt{k}_{\texttt{p}0}$*, two ciphertexts* C_1 *and* C_2*, and witnesses* r_1 *and* r_2*.*
- $\texttt{Hash}_0(\texttt{k}_{\texttt{h}1}, \texttt{k}_{\texttt{h}2}, L_{\widehat{\text{pw}}}, C_0)$ *computes hash value* h_0 *from hashing keys* $\texttt{k}_{\texttt{h}1}$ *and* $\texttt{k}_{\texttt{h}2}$ *and ciphertext* C_0*.*
- $\texttt{PHash}_0(\texttt{k}_{\texttt{p}_1}, \texttt{k}_{\texttt{p}_2}, L_{\widehat{\text{pw}}}, C_0, r_0)$ *computes hash value* h_0 *from projection keys* $\texttt{k}_{\texttt{p}_1}$ *and* $\texttt{k}_{\texttt{p}_2}$*, the ciphertext* C_0*, and witness* r_0*.*

A distributed SPHF protocol between three participants C, S_1, S_2 computing h_x and h_0 is described by three interactive protocols $\texttt{Setup}$, $\texttt{PHash}_x^D$ and $\texttt{Hash}_0^D$. Let Π denote D-SPHF as described above.

- $\texttt{Setup}(\text{pw}, \text{pw}_1, \text{pw}_2, C, S_1, S_2)$ initialises a new instance for each participant with (pw, C, S_1, S_2) for C, $(\text{pw}_1, S_1, C, S_2)$ for S_1 and $(\text{pw}_2, S_2, C, S_1)$ for S_2. Eventually, all participants compute and broadcast projection keys $\texttt{k}_{\texttt{p}_i}$ and encryptions $C_i \leftarrow \texttt{Enc}_{\texttt{pk}}^{\mathcal{L}}(\ell_i, \text{pw}_i; r_i)$ of their password (share) pw_i using $\Pi.\texttt{KGen}_{\texttt{H}}$, $\Pi.\texttt{KGen}_{\texttt{P}}$ and the associated encryption scheme $\mathcal{L}$. Participants store incoming $\texttt{k}_{\texttt{p}_i}, C_i$ for later use. After receiving $(\texttt{k}_{\texttt{p}_1}, C_1, \texttt{k}_{\texttt{p}_2}, C_2)$, the client computes $h_0 \leftarrow \Pi.\texttt{PHash}_0(\texttt{k}_{\texttt{p}_1}, \texttt{k}_{\texttt{p}_2}, L_{\widehat{\text{pw}}}, C_0, r_0)$ and $h_x \leftarrow \Pi.\texttt{Hash}_x(\texttt{k}_{\texttt{h}0}, L_{\widehat{\text{pw}}}, C_1, C_2)$.
- $\texttt{PHash}_x^D$ is executed between S_1 and S_2. Each server S_i performs $\texttt{PHash}_x^D$ on input $(\texttt{k}_{\texttt{p}_0}, \text{pw}_i, C_1, C_2, r_i)$ such that S_1 eventually holds h_x while S_2 learns nothing about h_x.
- $\texttt{Hash}_0^D$ is executed between S_1 and S_2. Each server S_i performs $\texttt{Hash}_0^D$ on input $(\text{pw}_i, \texttt{k}_{\texttt{h}i}, C_0, C_1, C_2)$ such that S_1 eventually holds h_0 while S_2 learns nothing about h_0.

We recall security and instantiation of D-SPHF in Appendix C

2.4 Ideal Functionalities

For our 2PAKE realisation we rely on some commonly used ideal functionalities within the UC framework. First, since we work in the crs model we require the crs functionality from [18], recalled in Figure 2 in Appendix D. We further need verified public keys on both servers. We use the ideal CA functionality $\mathcal{F}_{\texttt{CA}}$ from [17] for this, recalled in Figure 3 in Appendix D. Eventually, to establish unique query identifiers between the parties in a protocol run we use the $\mathcal{F}_{\texttt{init}}$ functionality from [7], recalled in Figure 4 in Appendix D.

3 Trapdoor Distributed Smooth Projective Hashing

T-SPHF enabled constructions of one-round UC-secure PAKE [10] because of simulatability even in presence of attackers who guess correct passwords. In order to use the trapdoor property for simulatability in 2PAKE protocols T-SPHF must first be extended to the distributed setting of D-SPHF (cf. Section 2.3). We denote this new flavor by TD-SPHF and describe it specifically for usage in our 2PAKE, i.e. using languages based on Cramer-Shoup ciphertexts. A more general description of TD-SPHF accounting for more servers and/or other languages can be obtained similarly to the general description of D-SPHF in [31].

Definition 5 (TD-SPHF). *Let* $L_{\widehat{\text{pw}}}$ *denote a language such that* $C = (C_0, C_1, C_2) \in L_{\widehat{\text{pw}}}$ *if there exists a witness* $r = (r_0, r_1, r_2)$ *proving so,* $\text{pw} = \text{pw}_1 + \text{pw}_2$ *and there exists a function* $\texttt{Dec}'$ *such that* $\texttt{Dec}'(C_1C_2) = \texttt{Dec}'(C_0)$*. A trapdoor distributed smooth projective hash function for language* $L_{\widehat{\text{pw}}}$ *consists of the following ten algorithms:*

- $(\texttt{crs}', \tau') \xleftarrow{R} \texttt{TSetup}(\texttt{crs})$ *generates* $\texttt{crs}'$ *with trapdoor* τ' *from* $\texttt{crs}$
- $\texttt{KGen}_{\texttt{H}}$*,* $\texttt{KGen}_{\texttt{P}}$*,* $\texttt{Hash}_x$*,* $\texttt{PHash}_x$*,* $\texttt{Hash}_0$*,* $\texttt{PHash}_0$ *behave as for D-SPHF*
- $b \leftarrow \texttt{VerKp}(\texttt{k}_{\texttt{p}}, L_{\widehat{\text{pw}}})$ *returns* $b = 1$ *iff* $\texttt{k}_{\texttt{p}}$ *is a valid projection key and* $b = 0$ *otherwise*
- $h_x \leftarrow \texttt{THash}_x(\texttt{k}_{\texttt{p}0}, L_{\widehat{\text{pw}}}, C_1, C_2, \tau')$ *computes hash value* h_x *of ciphertexts* C_1 *and* C_2 *using projection key* $\texttt{k}_{\texttt{p}0}$ *and trapdoor* τ'
- $h_0 \leftarrow \texttt{THash}_0(\texttt{k}_{\texttt{p}_1}, \texttt{k}_{\texttt{p}_2}, L_{\widehat{\text{pw}}}, C_0, \tau')$ *computes hash value* h_0 *of* C_0 *using projection keys* $\texttt{k}_{\texttt{p}_1}$ *and* $\texttt{k}_{\texttt{p}_2}$*, and trapdoor* τ'

Security of TD-SPHF can be derived from D-SPHF security and the extensions made on SPHF for T-SPHF. However, we do not consider security of TD-SPHF on its own but rather incorporate it in the security proof of the 2PAKE protocol in the following section. This is due to the fact that description of TD-SPHF is done only for this specific application such that a separate security definition is more distracting than giving any benefit. However, we define correctness and soundness of TD-SPHF since they differ from that of D-SPHF. In particular, *correctness* of TD-SPHF extends correctness of D-SPHF by the statement that for every valid ciphertext triple (C_0, C_1, C_2), generated by $\mathcal{L}$, and honestly generated keys $(\mathtt{k}_{\mathtt{h}0}, \mathtt{k}_{\mathtt{h}1}, \mathtt{k}_{\mathtt{h}2})$ and $(\mathtt{k}_{\mathtt{p}0}, \mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2})$, it holds not only that

$$\begin{aligned}&\mathtt{Hash}_0(\mathtt{k}_{\mathtt{h}1}, \mathtt{k}_{\mathtt{h}2}, L_{\widehat{\mathrm{pw}}}, C_0) = \mathtt{PHash}_0(\mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2}, L_{\mathrm{pw},\mathrm{pw}_1,\mathrm{pw}_2}, C_0, r_0) \text{ and}\\ &\mathtt{Hash}_x(\mathtt{k}_{\mathtt{h}0}, L_{\widehat{\mathrm{pw}}}, C_1, C_2) = \mathtt{PHash}_x(\mathtt{k}_{\mathtt{p}0}, L_{\mathrm{pw},\mathrm{pw}_1,\mathrm{pw}_2}, C_1, C_2, r_1, r_2)\end{aligned}$$

but also that $\mathtt{VerKp}(\mathtt{k}_{\mathtt{p}_i}, L_{\widehat{\mathrm{pw}}}) = 1$ for $i \in \{0,1,2\}$ and

$$\begin{aligned}&\mathtt{Hash}_0(\mathtt{k}_{\mathtt{h}1}, \mathtt{k}_{\mathtt{h}2}, L_{\widehat{\mathrm{pw}}}, C_0) = \mathtt{THash}_0(\mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2}, L_{\mathrm{pw},\mathrm{pw}_1,\mathrm{pw}_2}, C_0, \tau') \text{ and}\\ &\mathtt{Hash}_x(\mathtt{k}_{\mathtt{h}0}, L_{\widehat{\mathrm{pw}}}, C_1, C_2) = \mathtt{THash}_x(\mathtt{k}_{\mathtt{p}0}, L_{\mathrm{pw},\mathrm{pw}_1,\mathrm{pw}_2}, C_1, C_2, \tau').\end{aligned}$$

To capture soundness of TD-SPHFs we define (t,ε)-*soundness*, complementing the previous correctness extension, as follows.

Definition 6 (TD-SPHF (t,ε)-soundness). *Given* $\mathtt{crs}$, $\mathtt{crs}'$ *and* τ, *no adversary running in time at most* t *can produce a projection key* $\mathtt{k}_{\mathtt{p}}$, *a password* pw *with shares* pw_1 *and* pw_2, *a word* (C_0, C_1, C_2), *and valid witness* (r_0, r_1, r_2), *such that* $(\mathtt{k}_{\mathtt{p}0}, \mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2})$ *are valid, i.e.* $\mathtt{VerKp}(\mathtt{k}_{\mathtt{p}_i}, L_{\widehat{\mathrm{pw}}}) = 1$ *for* $i \in \{0,1,2\}$, *but*

$$\begin{aligned}&\mathtt{THash}_x(\mathtt{k}_{\mathtt{p}0}, L_{\widehat{\mathrm{pw}}}, C_1, C_2, \tau') \neq \mathtt{PHash}_x(\mathtt{k}_{\mathtt{p}0}, L_{\widehat{\mathrm{pw}}}, C_1, C_2, r_1, r_2) \text{ or}\\ &\mathtt{THash}_0(\mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2}, L_{\widehat{\mathrm{pw}}}, C_0, \tau') \neq \mathtt{PHash}_0(\mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2}, L_{\widehat{\mathrm{pw}}}, C_0, r_0)\end{aligned}$$

with probability at least $\varepsilon(\lambda)$. *The perfect soundness states that the property holds for any* t *and any* $\varepsilon(\lambda) > 0$.

3.1 Cramer-Shoup TD-SPHF

In the following we present TD-SPHF for labelled Cramer-Shoup ciphertexts by extending the corresponding D-SPHF from [31] with the trapdoor property from [10] in the setting of bilinear groups. Let $C = (\ell, u_1, u_2, e, v)$ denote a Cramer-Shoup ciphertext as defined in Section 2.

- $\mathtt{TSetup}(\mathtt{crs})$ draws a random $\tau' \in_R \mathbb{Z}_q$ and computes $\mathtt{crs}' = \zeta = g_2^{\tau'}$
- $\mathtt{KGen}_{\mathtt{H}}(L_{\widehat{\mathrm{pw}}})$ returns $\mathtt{k}_{\mathtt{h}i} = (\eta_{1,i}, \eta_{2,i}, \theta_i, \mu_i, \nu_i) \in_R \mathbb{Z}_p^{1\times 5}$ for $i \in \{0,1,2\}$
- $\mathtt{KGen}_{\mathtt{P}}(\mathtt{k}_{\mathtt{h}i}, L_{\widehat{\mathrm{pw}}})$ generates

$$\mathtt{k}_{\mathtt{p}_i} = (\mathtt{k}_{\mathtt{p}_{1,i}} = g_{1,1}^{\eta_{1,i}} g_{1,2}^{\theta_i} h^{\mu_i} c^{\nu_i}, \mathtt{k}_{\mathtt{p}_{2,i}} = g_{1,1}^{\eta_{2,i}} d^{\nu_i}, \mathtt{k}_{\mathtt{p}_{3,i}})$$

  with $\mathtt{k}_{\mathtt{p}_{3,i}} = (\chi_{1,1,i}, \chi_{1,2,i}, \chi_{2,i},\ \chi_{3,i}, \chi_{4,i})$ and

$$\chi_{1,1,i} = \zeta^{\eta_{1,i}}, \chi_{1,2,i} = \zeta^{\eta_{2,i}}, \chi_{2,i} = \zeta^{\theta_i}, \chi_{3,i} = \zeta^{\mu_i}, \chi_{4,i} = \zeta^{\nu_i} \text{ for } i \in \{0,1,2\}$$

- $\mathtt{Hash}_x(\mathtt{k}_{\mathtt{h}0}, L_{\widehat{\mathrm{pw}}}, C_1, C_2)$ computes

$$h'_x = (u_{1,1} \cdot u_{1,2})^{\eta_{1,0} + (\xi_1 + \xi_2)\eta_{2,0}} (u_{2,1} \cdot u_{2,2})^{\theta_0} ((e_1 \cdot e_2)/g_{1,1}^{\mathrm{pw}})^{\mu_0} (v_1 \cdot v_2)^{\nu_0}$$

  and returns $h_x = e(h'_x, g_2)$
- $\mathtt{PHash}_x(\mathtt{k}_{\mathtt{p}0}, L_{\widehat{\mathrm{pw}}}, C_1, C_2, r_1, r_2)$ computes $h'_x = \mathtt{k}_{\mathtt{p}_{1,0}}^{r_1+r_2} \mathtt{k}_{\mathtt{p}2,0}^{\xi_1 r_1 + \xi_2 r_2}$ and outputs $h_x = e(h'_x, g_2)$

– $\texttt{Hash}_0(\texttt{k}_{\texttt{h}1}, \texttt{k}_{\texttt{h}2}, L_{\widehat{\texttt{pw}}}, C_0)$ computes

$$h'_0 = u_{1,0}^{\eta_{1,1}+\eta_{1,2}+\xi_0(\eta_{2,1}+\eta_{2,2})} u_{2,0}^{\theta_1+\theta_2} (e_0/g_{1,1}^{\text{pw}})^{\mu_1+\mu_2} v_0^{\nu_1+\nu_2}$$

and outputs $h_0 = e(h'_0, g_2)$

– $\texttt{PHash}_0(\texttt{k}_{\texttt{p}1}, \texttt{k}_{\texttt{p}2}, L_{\widehat{\texttt{pw}}}, C_0, r_0)$ computes

$$h'_0 = (\texttt{k}_{\texttt{p}_{1,1}}\texttt{k}_{\texttt{p}_{1,2}})^{r_0} (\texttt{k}_{\texttt{p}_{2,1}}\texttt{k}_{\texttt{p}_{2,2}})^{r_0\xi_0}$$

and outputs $h_0 = e(h'_0, g_2)$

– $\texttt{VerKp}(\texttt{k}_{\texttt{p}_i}, L_{\widehat{\texttt{pw}}})$ verifies that

$$e(\texttt{k}_{\texttt{p}_{1,i}}, \texttt{crs}') \stackrel{?}{=} e(g_{1,1}, \chi_{1,1,i}) \cdot e(g_{1,2}, \chi_{2,i}) \cdot e(h, \chi_{3,i}) \cdot e(c, \chi_{4,i})$$

and

$$e(\texttt{k}_{\texttt{p}_{2,i}}, \texttt{crs}') \stackrel{?}{=} e(g_{1,1}, \chi_{1,2,i}) \cdot e(d, \chi_{4,i}) \text{ for } i \in \{0, 1, 2\}$$

– $\texttt{THash}_0(\texttt{k}_{\texttt{p}_1}, \texttt{k}_{\texttt{p}_2}, L_{\widehat{\texttt{pw}}}, C_0, \tau')$ computes

$$h_0 = \big[e(u_{1,0}, \chi_{1,1,1}\chi_{1,1,2}(\chi_{1,2,1}\chi_{1,2,2})^{\xi_0}) \cdot e(u_{2,0}, \chi_{2,1}\chi_{2,2})$$

$$\cdot e(e_0/g_{1,1}^{\text{pw}}, \chi_{3,1}\chi_{3,2}) \cdot e(v_0, \chi_{4,1}\chi_{4,2})\big]^{1/\tau'}$$

– $\texttt{THash}_x(\texttt{k}_{\texttt{p}_0}, L_{\widehat{\texttt{pw}}}, C_1, C_2, \tau')$ computes

$$h_x = \big[e(u_{1,1}u_{1,2}, \chi_{1,1,0}\chi_{1,2,0}^{\xi_1+\xi_2}) \cdot e(u_{2,1}u_{2,2}, \chi_{2,0}) \cdot e((e_1e_2)/g_{1,1}^{\text{pw}}, \chi_{3,0})$$

$$\cdot e(v_1v_2, \chi_{4,0})\big]^{1/\tau'}$$

Distributed computation of $\texttt{PHash}_x$ and $\texttt{Hash}_0$ is done as in D-SPHF with additional proofs for correctness and adding the pairing computation at the end to lift the hash value into $\mathbb{G}_T$. We formalise execution of the Cramer-Shoup TD-SPHF in the following paragraph. Necessary zero-knowledge proofs are described in the subsequent two paragraphs and only referenced in the description of the TD-SPHF. We describe the Σ protocol here, which we can use after transforming it to a committed Σ protocol (cf. Section 2). Note that we merge $\texttt{crs}$ and $\texttt{crs}'$ here for readability. Protocol participants are denoted C, S_1 and S_2 if their role is specified, or P, Q and R otherwise. Let further 0 denote the client's index and 1, 2 the indices of servers S_1, S_2, respectively. The session ID is given by $\texttt{sid} = C||S_1||S_2$ and the unique query identifier $\texttt{qid}$ is agreed upon start using $\mathcal{F}_{\texttt{init}}$.

All TD-SPHF participants have $\texttt{crs} = (q, g_{1,1}, g_{1,2}, h, c, d, \mathbb{G}_1, g_2, \zeta, \mathbb{G}_2, \mathbb{G}_T, e, H_k)$ as common input where $\tau = (x_1, x_2, y_1, y_2, z)$ is the $\texttt{crs}$ trapdoor, i.e. the according Cramer-Shoup secret key, and τ' the trapdoor, i.e. discrete logarithm to base g_2, of $\texttt{crs}' = \zeta$. Each server holds an ElGamal key pair $(\texttt{pk}_1, \texttt{dk}_1)$ and $(\texttt{pk}_2, \texttt{dk}_2)$ respectively such that $\texttt{pk}_1$ is registered with the $\texttt{CA}$ for S_1 and $\texttt{pk}_2$ for S_2 and thus available to all parties (using $\mathcal{F}_{\texttt{CA}}$). An, otherwise unspecified, protocol participant P is initiated with $(\texttt{NS}, \texttt{sid}, \texttt{qid}, P, x)$. We further define $\text{pw}_0 = \text{pw}$.

CS TD-SPHF Computation

a) Generate TD-SPHF keys $\texttt{k}_{\texttt{h}i} \in_R \mathbb{Z}_q^5$ and $\texttt{k}_{\texttt{p}_i} = (\texttt{k}_{\texttt{p}_{1,i}} = g_{1,1}^{\eta_{1,i}} g_{1,2}^{\theta_i} h^{\mu_i} c^{\nu_i}, \texttt{k}_{\texttt{p}_{2,i}} = g_{1,1}^{\eta_{2,i}} d^{\nu_i}, \chi_{1,1,i} = \zeta^{\eta_{1,i}}, \chi_{1,2,i} = \zeta^{\eta_{2,i}}, \chi_{2,i} = \zeta^{\theta_i}, \chi_{3,i} = \zeta^{\mu_i}, \chi_{4,i} = \zeta^{\nu_i})$. Encrypt pw_i to $C = (\ell_i, u_{1,i}, u_{2,i}, e_i, v_i) \leftarrow (\ell, g_{1,1}^{r_i}, g_{1,2}^{r_i}, h^{r_r} g_{1,1}^{\text{pw}_i}, (cd^{\xi_i})^{r_i})$ with $\xi_i = H_k(\ell_i, u_{1,i}, u_{2,i}, e_i)$ for $\ell_i = \texttt{sid}||\texttt{qid}||\texttt{k}_{\texttt{p}_i}$ and $r_i \in_R \mathbb{Z}_q$. If $P = S_1$, set $h_0 = h_x = \texttt{null}$. Output $(\texttt{sid}, \texttt{qid}, 0, P, C_i, \texttt{k}_{\texttt{p}_i})$ to Q and R.

b) When P, waiting for the initial messages, is receiving a message $(\mathtt{sid}, \mathtt{qid}, 0, Q, C_1, \mathtt{k}_{\mathtt{p}_1})$ and $(\mathtt{sid}, \mathtt{qid}, 0, R, C_2, \mathtt{k}_{\mathtt{p}_2})$ it proceeds as follows. P proceeds only if the projection keys $\mathtt{k}_{\mathtt{p}_1}$ and $\mathtt{k}_{\mathtt{p}_2}$ are correct, i.e. $\mathtt{VerKp}(\mathtt{k}_{\mathtt{p}_1}, L_{\widehat{\mathrm{pw}}}) = 1$ and $\mathtt{VerKp}(\mathtt{k}_{\mathtt{p}_2}, L_{\widehat{\mathrm{pw}}}) = 1$. If the verification fails, P outputs $(\mathtt{sid}, \mathtt{qid}, \perp, \perp)$ and aborts the protocol.

i) If $P = C$, compute
$h_x = e((u_{1,1} \cdot u_{1,2})^{\eta_{1,0}+(\xi_1+\xi_2)\eta_{2,0}}(u_{2,1} \cdot u_{2,2})^{\theta_0}$
$((e_1 \cdot e_2)/g_{1,1}^{\mathrm{pw}})^{\mu_0}(v_1 \cdot v_2)^{\nu_0}, g_2)$ and
$h_0 = e\left((\mathtt{k}_{\mathtt{p}_{1,1}}\mathtt{k}_{\mathtt{p}_{1,2}})^{r_0}(\mathtt{k}_{\mathtt{p}_{2,1}}\mathtt{k}_{\mathtt{p}_{2,2}})^{r_0\xi_0}, g_2\right)$, and outputs $(\mathtt{sid}, \mathtt{qid}, h_0, h_x)$.

ii) If $P = S_2$, compute $h_{x,2} = (\mathtt{k}_{\mathtt{p}_{1,0}} \cdot \mathtt{k}_{\mathtt{p}_{2,0}}^{\xi_2})^{r_2}$ and $C_{h_{x,2}} = g_{1,1}^{H(h_{x,2}, \mathtt{Co}_1)} h^{r_{c1}}$ with $r_{c1} \in_R \mathbb{Z}_q$ and send $(\mathtt{sid}, \mathtt{qid}, \mathtt{PHash}_x, 0, S_2, C_{h_{x,2}})$ to S_1.

iii) If $P = S_1$, compute $m_0 = \mathtt{Enc}_{\mathtt{pk}_1}^{\mathrm{EG}}(g_{1,1}^{-\mu_1}; r)$ and $c_0 = \mathtt{Enc}_{\mathtt{pk}_1}^{\mathrm{EG}}(g_{1,1}^{\mathrm{pw}_1}; r')$ with $r, r' \in_R \mathbb{Z}_q$, and send $(\mathtt{sid}, \mathtt{qid}, \mathtt{Hash}_0, 0, S_1, m_0, c_0)$ to S_2.

c) On input $(\mathtt{sid}, \mathtt{qid}, \mathtt{PHash}_x, 0, S_2, C_{h_{x,2}})$ S_1 in the correct state draws challenge $\mathfrak{c} \in_R \mathbb{Z}_q$ and returns $(\mathtt{sid}, \mathtt{qid}, \mathtt{PHash}_x, 1, S_1, \mathfrak{c})$ to S_2.

d) On input $(\mathtt{sid}, \mathtt{qid}, \mathtt{PHash}_x, 1, S_1, \mathfrak{c})$ S_2 in the correct state computes $C_{s_{h_{x,2}}} = g_{1,1}^{H(\mathtt{Rs}_1)} h^{r_{c2}}$ with $r_{c2} \in_R \mathbb{Z}_q$ and sends $(\mathtt{sid}, \mathtt{qid}, \mathtt{PHash}_x, 2, S_2, C_{s_{h_{x,2}}})$ to S_1. Subsequently, it sends $(\mathtt{sid}, \mathtt{qid}, \mathtt{PHash}_x, 3, S_2, h_{x,2}, \mathtt{Co}_1, \mathtt{Rs}_1, r_{c1}, r_{c2})$ to S_1.

e) On input $(\mathtt{sid}, \mathtt{qid}, \mathtt{PHash}_x, 2, S_2, C_{s_{h_{x,2}}})$ S_1 in the correct state stores it and waits for the final $\mathtt{PHash}_x$ message.

f) On input $(\mathtt{sid}, \mathtt{qid}, \mathtt{PHash}_x, 3, S_2, h_{x,2}, \mathtt{Co}_1, \mathtt{Rs}_1, r_{c1}, r_{c2})$ S_1 in the correct state parses $\mathtt{Co}_1$ as (t_1, t_2) and $\mathtt{Rs}_2$ as $s_{h_{x,2}}$ and verifies correctness of commitments and the ZKP and computes $h_x = e(h_{x,2} \cdot (\mathtt{k}_{\mathtt{p}_{0,1}} \cdot \mathtt{k}_{\mathtt{p}_{0,2}}^{\xi_1})^{r_1}, g_2)$ if the verifications are successful, $h_x \neq \perp$ and $h_0 \neq \perp$, or sets $h_0 = \perp$ and $h_x = \perp$ otherwise.

g) On input $(\mathtt{sid}, \mathtt{qid}, \mathtt{Hash}_0, 0, S_1, m_0, c_0)$ S_2 in the correct state retrieves $\mathtt{pk}_1$ from $\mathcal{F}_{\mathtt{CA}}$ and computes $C_{\mathtt{Hash}_{0,1}} = g_{1,1}^{H(m_1, m_2, \mathtt{Co}_2)} h^{r_{c3}}$ with $r_{c3} \in_R \mathbb{Z}_q$, $m_1 \leftarrow m_0^{\mathrm{pw}_2} \times c_0^{-\mu_2} \times \mathtt{Enc}_{\mathtt{pk}_1}^{\mathrm{EG}}(g_{1,1}^{-\mu_2 \cdot \mathrm{pw}_2} \cdot u_{1,0}^{\eta_{1,2}+\xi_0\eta_{2,2}} \cdot u_{2,0}^{\theta_2} \cdot e_0^{\mu_2} \cdot v_0^{\nu_2}; r'')$, and $m_2 \leftarrow \mathtt{Enc}_{\mathtt{pk}_1}^{\mathrm{EG}}(g_{1,1}^{-\mu_2}; r''')$ with $r'', r''' \in \mathbb{Z}_q$, and sends $(\mathtt{sid}, \mathtt{qid}, \mathtt{Hash}_{0,1}, S_2, C_{\mathtt{Hash}_{0,1}})$ back to S_1.

h) On input $(\mathtt{sid}, \mathtt{qid}, \mathtt{Hash}_{0,1}, S_2, C_{\mathtt{Hash}_{0,1}})$ S_1 in the correct state draws challenge $\mathfrak{c} \in_R \mathbb{Z}_q$ and returns $(\mathtt{sid}, \mathtt{qid}, \mathtt{Hash}_{0,2}, S_1, \mathfrak{c})$ to S_2.

i) On input $(\mathtt{sid}, \mathtt{qid}, \mathtt{Hash}_{0,2}, S_1, \mathfrak{c})$ S_2 in the correct state computes $C_{\mathtt{Rs2}} = g_{1,1}^{H(\mathtt{Rs}_2)} h^{r_{c4}}$ with $r_{c4} \in_R \mathbb{Z}_q$ and sends $(\mathtt{sid}, \mathtt{qid}, \mathtt{Hash}_{0,3}, S_2, C_{\mathtt{Rs2}})$ to S_1. Subsequently, it sends $(\mathtt{sid}, \mathtt{qid}, \mathtt{Hash}_{0,4}, S_2, m_1, m_2, \mathtt{Co}_2, \mathtt{Rs}_2, r_{c3}, r_{c4})$ to S_1.

j) On input $(\mathtt{sid}, \mathtt{qid}, \mathtt{Hash}_{0,4}, S_2, m_1, m_2, \mathtt{Co}_2, \mathtt{Rs}_2, r_{c3}, r_{c4})$ S_1 in the correct state parses $\mathtt{Co}_2$ as $(t_{\overline{m}1}, t_{\overline{m}2}, t_{e2}, t_{v2}, t_{\mathtt{k}_{\mathrm{p}}12}, t_{\mathtt{k}_{\mathrm{p}}22})$ and $\mathtt{Rs}_2$ as $(s_{\mathrm{pw}_2}, s_{\mu 2}, s_{\eta 12}, s_{\eta 22}, s_{\theta 2}, s_{\nu 2}, s_{r2})$, verifies correctness of commitments and ZKP, and computes $h_0 = e\left(g_{1,1}^{-\mu_1 \cdot \mathrm{pw}_1} \cdot \mathtt{Dec}_{\mathtt{dk}_1}^{\mathrm{EG}}(m_1) \cdot u_{1,0}^{\eta_{1,1}+\xi_0\eta_{2,1}} \cdot u_{2,0}^{\theta_1} \cdot e_0^{\mu_1} \cdot v_0^{\nu_1}, g_2\right)$ if the verifications are successful, $h_x \neq \perp$ and $h_0 \neq \perp$, or sets $h_0 = \perp$ and $h_x = \perp$.

k) Eventually S_1 outputs $(\mathtt{sid}, \mathtt{qid}, h_0, h_x)$ if $h_0 \neq \mathtt{null}$ and $h_x \neq \mathtt{null}$.

ZK Proof for $\mathtt{PHash}_x$ *Correctness* In order to ensure correct computation of h_x on S_1 server S_2 has to prove correctness of his computations. To this end S_2 sends, in addition to the $\mathtt{PHash}_x$ message $h_{x,2}$ the following zero-knowledge proof.

$$\mathsf{ZKP}\{(r_2): \quad h_{x,2} = (\mathtt{k}_{\mathtt{p}_{1,0}}\mathtt{k}_{\mathtt{p}_{2,0}}^{\xi_2})^{r_2} \ \wedge \ v_2 = (cd^{\xi_2})^{r_2}\} \tag{1}$$

where r_2 is the randomness used to create C_2, ξ_2 and v_2 are part of C_2, $\mathtt{k}_{\mathtt{p}_{1,0}}, \mathtt{k}_{\mathtt{p}_{2,0}}$ are part of C's projection key, and c, d are from the $\mathtt{crs}$. The construction of the according zero-knowledge proof is straight-forward. The prover computes commitments

$$t_{hx2} = (\mathtt{k}_{\mathtt{p}_{1,0}}\mathtt{k}_{\mathtt{p}_{2,0}}^{\xi_2})^{k_{hx2}}; \quad t_{v2} = (cd^{\xi_2})^{k_{hx2}}$$

with fresh randomness $k_{hx2} \in_R \mathbb{Z}_q$, and response $s_{r2} = k_{hx2} - \mathfrak{c} r_2$ for verifier provided challenge $\mathfrak{c}$. This allows the verifier to check

$$t_{hx2} \stackrel{?}{=} h_{x,2}^{\mathfrak{c}} (\mathtt{k}_{\mathtt{p}1,0} \mathtt{k}_{\mathtt{p}2,0}^{\xi_2})^{s_{hx2}}; \quad t_{v2} \stackrel{?}{=} v_2^{\mathfrak{c}} (cd^{\xi_2})^{s_{hx2}}.$$

It is easy to see that this zero-knowledge proof is correct, sound and (honest-verifier) simulatable. We refer to the messages as $\mathtt{Co}_1 = (t_{hx2}, t_{v2})$, $\mathtt{Rs}_1 = s_{r2}$, and $\mathtt{Ch}_1 = \mathfrak{c}$.

ZK Proof for $\mathtt{Hash}_0$ *Correctness* Let $\overline{m}_1$ and $\overline{m}_2$ denote the messages encrypted in m_1 and m_2 respectively and $m_{0,1}$ and $c_{0,1}$ the second part (e) of the ElGamal ciphertext m_0, c_1 respectively. In order to ensure correct computation of h_0 on S_1 server S_2 has to prove correctness of his computations. To this end S_2 sends, additionally to the $\mathtt{Hash}_0$ messages $\overline{m}_1$ and $\overline{m}_2$ the following zero-knowledge proof

$$\begin{aligned} \mathsf{ZKP}\big\{(x, \eta_{1,2}, \eta_{2,2}, \theta_2, \mu_2, \nu_2, r_2) : \overline{m}_1 &= m_{0,1}^{\mathrm{pw}_2} c_{0,1}^{-\mu_2} g_{1,1}^{-\mu_2 x} u_{1,0}^{\eta_{1,2}+\xi_0\eta_{2,2}} u_{2,0}^{\theta_2} e_0^{\mu_2} v_0^{\nu_2} \\ \wedge\ \overline{m}_2 &= g_{1,1}^{-\mu_2} \ \wedge\ e_2 = h^{r_2} g_{1,1}^{\mathrm{pw}_2} \ \wedge\ v_2 = (cd^{\xi_2})^{r_2} \\ \wedge\ \mathtt{k}_{\mathtt{p}1,2} &= g_{1,1}^{\eta_{1,2}} g_{1,2}^{\theta_2} h^{\mu_2} c^{\nu_2} \ \wedge\ \mathtt{k}_{\mathtt{p}2,2} = g_{1,1}^{\eta_{2,2}} d^{\nu_2}\big\}, \end{aligned} \tag{2}$$

where r_2 is the randomness used to create C_2, ξ_2 and v_2 are part of C_2, ξ_0 is part of C_0, $(\mu_2, \eta_{1,2}, \eta_{2,2}, \theta_2, \nu_2)$ is S_2's hashing key, pw_2 S_2's password share, and c, d are from the $\mathtt{crs}$. The construction of the according Σ proof is straight-forward. The prover computes commitments

$$\begin{aligned} &t_{\overline{m}1} = m_{0,1}^{\mathrm{pw}_2} c_{0,1}^{k_{\mu 2}} \overline{m}_2^{k_x} u_{1,0}^{k_{\eta 12}+\xi_0 k_{\eta 22}} u_{2,0}^{k_{\theta 2}} e_0^{-k_{\mu 2}} v_0^{k_{\nu 2}}; \quad t_{\overline{m}2} = g_{1,1}^{k_{\mu 2}}; \quad t_{e2} = h^{k_{r2}} g_{1,1}^{\mathrm{pw}_2}; \\ &t_{v2} = (cd^{\xi_2})^{k_{r2}}; \quad t_{\mathtt{k}_\mathtt{p}12} = g_{1,1}^{k_{\eta 12}} g_{1,2}^{k_{\theta 2}} h^{k_{\mu 2}} c^{k_{\nu 2}}; \quad t_{\mathtt{k}_\mathtt{p}22} = g_{1,1}^{k_{\eta 22}} d^{k_{\nu 2}} \\ &\text{for} \quad k_{\mathrm{pw}_2}, k_{\mu 2}, k_{\eta 12}, k_{\eta 22}, k_{\theta 2}, k_{\nu 2} \in_R \mathbb{Z}_q \end{aligned}$$

and responses

$$\begin{aligned} &s_{\mathrm{pw}_2} = k_{\mathrm{pw}_2} - \mathfrak{c}\mathrm{pw}_2; \quad s_{\mu 2} = k_{\mu 2} + \mathfrak{c}\mu_2; \quad s_{\eta 12} = k_{\eta 12} - \mathfrak{c}\eta_{1,2}; \quad s_{\eta 22} = k_{\eta 22} - \mathfrak{c}\eta_{2,2}; \\ &s_{\theta 2} = k_{\theta 2} - \mathfrak{c}\theta_2; \quad s_{\nu 2} = k_{\nu 2} - \mathfrak{c}\nu_2; \quad s_{r2} = k_{r2} - \mathfrak{c} r_2 \end{aligned}$$

for verifier provided challenge $\mathfrak{c}$. This allows the verifier to check

$$\begin{aligned} &t_{\overline{m}1} \stackrel{?}{=} \overline{m}_1^{\mathfrak{c}} m_{0,1}^{s_{\mathrm{pw}_2}} c_{0,1}^{s_{\mu 2}} \overline{m}_2^{s_{\mathrm{pw}_2}} u_{1,0}^{s_{\eta 12}+\xi_0 s_{\eta 22}} u_{2,0}^{s_{\theta 2}} e_0^{s_{\mu 2}} v_0^{s_{\nu 2}}; \quad t_{\overline{m}2} \stackrel{?}{=} \overline{m}_2^{\mathfrak{c}} g_{1,1}^{s_{\mu 2}}; \quad t_{e2} \stackrel{?}{=} e_2^{\mathfrak{c}} h^{s_{r2}} g_{1,1}^{s_{\mathrm{pw}_2}}; \\ &t_{v2} \stackrel{?}{=} v_2^{\mathfrak{c}} (cd^{\xi_2})^{s_{r2}}; \quad t_{\mathtt{k}_\mathtt{p}12} \stackrel{?}{=} \mathtt{k}_{\mathtt{p}1,2}^{\mathfrak{c}} g_{1,1}^{s_{\eta 12}} g_{1,2}^{s_{\theta 2}} h^{s_{\mu 2}} c^{s_{\nu 2}}; \quad t_{\mathtt{k}_\mathtt{p}22} \stackrel{?}{=} \mathtt{k}_{\mathtt{p}2,2}^{\mathfrak{c}} g_{1,1}^{s_{\eta 22}} d^{s_{\nu 2}}. \end{aligned}$$

While this is mainly a standard zero-knowledge proof $t_{\overline{m}1}$ uses $\overline{m}_2$ instead of $g_{1,1}$ as base for the third factor and k_{pw_2} as exponent (s_{pw_2} in the verification). This is necessary due to the fact that the exponent $-\mu_2 \mathrm{pw}_2$ of the third factor in $\overline{m}_1$ is a product of two values that have to be proven correct. The ZK proof uses the auxiliary message $\overline{m}_2$ to prove that $\log_{g_{1,1}}(\overline{m}_2) = -\mu_2$ such that it is sufficient to prove $\log_{\overline{m}_2}(\overline{m}_2^{\mathrm{pw}_2}) = \mathrm{pw}_2$. We refer to the messages as $\mathtt{Co}_2 = (t_{\overline{m}1}, t_{\overline{m}2}, t_{e2}, t_{v2}, t_{\mathtt{k}_\mathtt{p}12}, t_{\mathtt{k}_\mathtt{p}22})$, $\mathtt{Rs}_2 = (s_{\mathrm{pw}_2}, s_{\mu 2}, s_{\eta 12}, s_{\eta 22}, s_{\theta 2}, s_{\nu 2}, s_{r2})$, and $\mathtt{Ch}_2 = \mathfrak{c}$.

4 Universally Composable Two-Server PAKE

With TD-SPHF it is straight forward to build a 2PAKE protocol. We follow the general framework described in [31] to build 2PAKE protocols from distributed smooth projective hash functions. However, instead of aiming for key generation, where the client establishes a key with each of the two servers, we focus on a protocol that establishes a single key with one server, w.l.o.g. the first server. By running the protocol twice, keys can be exchanged between the client and the second sever. Note that UC security allows concurrent execution of the protocol such that round complexity is not increased by establishing two keys.

4.1 The Protocol

We obtain our 2PAKE protocol using the general 2PAKE framework from [31] yet using our TD-SPHF instead of original D-SPHF. Client C and both servers S_1 and S_2 execute a TD-SPHF protocol from Section 3 which provides C and S_1 with two hash values h_0 and h_x each. The session key is then computed by both as a product $\mathtt{sk} = h_0 \cdot h_x$.

4.2 Ideal Functionality for 2PAKE

Our ideal functionality for 2PAKE with implicit client authentication, $\mathcal{F}_{\text{2PAKE}}$, is given in Figure 1. Observe that implicit client authentication is sufficient for building UC-secure channels [20]. The ideal adversary can take control of any server from the outset of the protocol and learn the corresponding password share. The actual password remains hidden unless the adversary corrupts both servers. The use of static corruptions is motivated in the following. First, as explained in [19], PAKE security against static corruptions in the UC model implies security against adaptive corruptions in the BPR model. Second, existing single-server PAKE protocols that are UC-secure against adaptive corruptions, e.g. [1,2,3], rely on more complex SPHF constructions that are not translatable to the distributed setting of D-SPHF. We discuss the relation between our new UC formalisation of 2PAKE and the known BPR-based security model in Appendix E.

Functionality $\mathcal{F}_{\text{2PAKE}}$

The functionality $\mathcal{F}_{\text{2PAKE}}$ is parameterised by a security parameter λ. It interacts with an adversary, a client C and two servers S_1 and S_2 via the following interfaces. Without loss of generality the key is exchanged between C and S_1.

KEX Init$_C$: Upon input $(\mathtt{KEXinit}, \mathtt{sid}, \mathtt{qid}, \text{pw})$ from client C, check that $\mathtt{sid}$ is (C, S_1, S_2) and that $\mathtt{qid}$ is unique (entries $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, S_1, \alpha_1)$ or $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, S_2, \alpha_2)$ may exist) and send $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, C)$ to $\mathtt{SIM}$. If this is a valid request, create a *fresh* record $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, C, \text{pw})$.

KEX Init$_S$: Upon input $(\mathtt{KEXinit}, \mathtt{sid}, \mathtt{qid}, \alpha_b)$ from server S_b, $b \in \{1, 2\}$, check that $\mathtt{sid}$ is (C, S_1, S_2) and that $\mathtt{qid}$ is unique (entries $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, C, \text{pw})$ or $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, S_{3-b}, \alpha_{3-b})$ may exist) and send $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, S_b)$ to $\mathtt{SIM}$. If this is a valid request, create a fresh record $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, S_b, \alpha_b)$.

TestPwd: Upon input $(\mathtt{TP}, \mathtt{sid}, \mathtt{qid}, \text{pw}')$ from $\mathtt{SIM}$ check that a fresh record $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, C, \text{pw})$ exists. If this is the case, mark $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, S_1, \alpha_1)$ as $\mathtt{compromised}$ and reply with "correct guess" if $\text{pw} = \text{pw}'$, and mark it as $\mathtt{interrupted}$ and reply with "wrong guess" if $\text{pw} \neq \text{pw}'$.

Failed: Upon input $(\mathtt{FA}, \mathtt{sid}, \mathtt{qid})$ from $\mathtt{SIM}$ check that records $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, C, \text{pw})$ and $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, S_1, \alpha_1)$ exist that are not marked $\mathtt{completed}$. If this is the case, mark both as $\mathtt{failed}$.

NewKey: Upon input $(\mathtt{NK}, \mathtt{sid}, \mathtt{qid}, P, \mathtt{sk}')$ from $\mathtt{SIM}$ with $P \in \{C, S_1\}$, check that a respective $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, C, \text{pw})$ or $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, S, \alpha_1)$ record exists, $\mathtt{sid} = (C, S_1, S_2)$, $|\mathtt{sk}'| = \lambda$, then:

- If the session is $\mathtt{compromised}$, or either C or S_1 and S_2 are corrupted, then output $(\mathtt{NK}, \mathtt{sid}, \mathtt{qid}, \mathtt{sk}')$ to P; else
- if the session is *fresh* and a key $\mathtt{sk}$ was sent to P' with $\mathtt{sid} = (P, P', S_2)$ or $\mathtt{sid} = (P', P, S_2)$ while $(\mathtt{KEX}, \mathtt{sid}, \mathtt{qid}, P', \cdot)$ was fresh, then output $(\mathtt{NK}, \mathtt{sid}, \mathtt{qid}, \mathtt{sk})$ to P.
- In any other case, pick a new random key $\mathtt{sk}$ of length λ, and send $(\mathtt{NK}, \mathtt{sid}, \mathtt{qid}, \mathtt{sk})$ to P.

In any case, mark $\mathtt{qid}$ as $\mathtt{completed}$ for P.

Fig. 1: Ideal Functionality $\mathcal{F}_{\text{2PAKE}}$

2PAKE Functionality Our $\mathcal{F}_{\text{2PAKE}}$ is very similar to single-server PAKE functionality but assumes two servers from which one generates a session key. The main difference is in the modelling of participants. We

specify two initialisation interfaces **KEX Init**, one for the client and one for the servers. A client is initialised with a password pw while a server gets a password share α_b. The **TestPwd** interface allows the ideal world adversary to test client passwords. A tested session is marked `interrupted` if the guess is wrong, i.e. client and server in this session receive randomly chosen, independent session keys, or marked as `compromised` if the password guess is correct, i.e. the attacker is now allowed to set the session key. The attacker can only test client passwords but not password shares of the servers. Without knowledge of the password or any password share, a share is a uniformly at random chosen element and therefore not efficiently guessable. If the adversary corrupted server S_2, retrieving the second password share α_1 from S_1 is equivalent to guessing the password. Complementing the TestPwd interface is a **Failed** interface that allows the adversary to let sessions fail. This allows the attacker to prevent protocol participants from computing any session, i.e. failed parties do not compute a session key. Eventually the **NewKey** interface generates session keys for client C and server S_1. NewKey calls for S_2 are ignored. If client C or server S_1 and S_2 are corrupted, or the attacker guessed the correct password, the adversary chooses the session key. If a session key was chosen for the partnered party and the session was fresh at that time, i.e. not `compromised` or `interrupted`, the same session key is used again. In any other case a new random session key is drawn.

Instead of using a single session identifier `sid` we use `sid` and `qid`. The session identifier `sid` is composed of the three participants (C, S_1, S_2) (note that we use the client C also as "username" that identifies its account on the servers) and therefore human memorable and unique. To handle multiple, concurrent 2PAKE executions of one `sid`, we use a query identifier `qid` that is unique within `sid` and can be established with $\mathcal{F}_{\texttt{init}}$. In the multi-session extension $\widehat{\mathcal{F}}_{\text{2PAKE}}$ the `sid` becomes `ssid` and `sid` is a globally unique identifier for the used universe, i.e. server public keys (`CA`) and `crs`.

4.3 Security

The following theorem formalises the security of the proposed 2PAKE protocol. Note that we do not rely on any security of the TD-SPHF. Instead we reduce the security of our 2PAKE protocol directly to the underlying problem (SXDH). Thereby, we give an indirect security proof of the proposed TD-SPHF.

Theorem 1. *The 2PAKE protocol from Section 4.1 securely realises $\widehat{\mathcal{F}}_{\text{2PAKE}}$ with static corruptions in the $\mathcal{F}_{\texttt{crs}}$-$\mathcal{F}_{\texttt{CA}}$-hybrid model if the DDH assumption holds in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$ and if H_k is a universal one-way hash function.*

Sequence of Games We start the proof of Theorem 1 by giving a sequence of games with $\mathcal{G}_1$ equal to the real-world execution with honest participants following the protocol description and the real-world adversary $\mathcal{A}$ that may have control over a set of participants, and $\mathcal{G}_{17}$ equal to the ideal-world execution where the protocol is replaced with the ideal functionality $\mathcal{F}_{\text{D-SPHF}}$ acting on behalf of all honest protocol participants and the ideal-world adversary `SIM`, detailed later. Let $\texttt{view}_i$ denote the view of environment $\mathcal{Z}$ when interacting with game $\mathcal{G}_i$. Note that `view` is implicitly parametrised with `sid` and the security parameter λ. Security then follows from showing that each $\texttt{view}_i$ is computationally indistinguishable from the subsequent $\texttt{view}_{i+1}$, such that we can eventually follow by an hybrid argument that $\texttt{view}_1$ and $\texttt{view}_{17}$ are computationally indistinguishable and the protocol therefore securely realises the ideal functionality $\mathcal{F}_{\text{2PAKE}}$. All participants in the games are operated by the challenger $\mathcal{C}$ (receiving the participants input from environment $\mathcal{Z}$), which we modify from game to game. Every session for an $\texttt{sid} = (C, S_1, S_2)$ is started with a KexInit call for each participant, defining secrets, roles, and the used query identifier. Invalid messages, i.e. messages that do not pass the usual tests such as group membership, are discarded by the challenger. Note that we usually only give the actual payload of messages and omit additional parts such as `sid`, `qid` etc.

$\mathcal{G}_1$: Game 1 is the real-world experiment in which $\mathcal{Z}$ interacts with real participants that follow, if honest, the protocol description, and the real-world adversary $\mathcal{A}$ controlling the corrupted parties. All participants are honestly simulated by challenger $\mathcal{C}$ that knows all their inputs.

$\mathcal{G}_2$ **:** This game is identical to $\mathcal{G}_1$, except that the $\mathtt{crs}$ is generated by $\mathcal{C}$ such that it knows the trapdoor τ. Note that the second trapdoor τ' for ζ is *not* controlled by $\mathcal{C}$ yet as this would destroy any security. Knowledge of τ allows $\mathcal{C}$ to decrypt ciphertexts C_i and retrieve the used message. This does not change anything and is therefore perfectly indistinguishable from $\mathcal{G}_1$.

$\mathcal{G}_3$ **:** When $\mathcal{C}$, on behalf of S_1, receives first messages $(C_0, \mathtt{k}_{\mathtt{p}_0})$ and $(C_2, \mathtt{k}_{\mathtt{p}_2})$, it decrypts C_0 to pw' and checks if this is the correct password, i.e. $\mathrm{pw}' = \mathrm{pw}$. If this is not the case, $\mathrm{pw}' \neq \mathrm{pw}$, $\mathcal{C}$ chooses a random $h'_0 \in_R \mathbb{G}_T$ if the subsequent $\mathtt{Hash}_0$ computation with S_2 is successful, i.e. all zero-knowledge proofs can be verified, and aborts S_1 otherwise. We claim that $\mathtt{view}_2$ is computationally indistinguishable from $\mathtt{view}_3$. The probability to distinguish the two games is bounded by the negligible probability to notice that h_0 is now chosen uniformly at random. Since $C = (C_0, C_1, C_2)$ is not in $L_{\mathrm{pw},\mathrm{pw}_1,\mathrm{pw}_2}$ the computation of $\mathtt{Hash}_0$ between S_1 and S_2 yields a uniformly at random distributed hash value h_0. This can be either deduced from the smoothness proven for the generic (not distributed) T-SPHF in [11] or by the following simplified argument. As long as $C \notin L_{\mathrm{pw},\mathrm{pw}_1,\mathrm{pw}_2}$ the same argument as used for SPHF and D-SPHF can be used, namely that h_0 is linearly independent from the adversarially known values and therefore indistinguishable from a random one. However, this is not sufficient in this case as the attacker has the possibility to distinguish real h_0 values from random ones with use of the third projection keys $\mathtt{k}_{\mathtt{p}3,i}$. To show that this is not possible we show how to break the DDH assumption in $\mathbb{G}_2$ if there exists a distinguisher that can distinguish real h_0 from random ones. To this end we build a DDH triple $(\zeta, \mathfrak{a}, \mathfrak{b})$ with $\mathtt{crs}' = \zeta = g_2^{\tau'}$ as follows. Let $\mathfrak{a} = \zeta^{\alpha}$ and $\mathfrak{b} = g_2^{\alpha}$, then $(\zeta, \mathfrak{a}, \mathfrak{b})$ is obviously a DDH triple. To link this to the TD-SPHF we set $\alpha = \mathtt{k}_{\mathtt{h}i,j}$, then $\mathfrak{a} = \mathtt{k}_{\mathtt{p}3,i,j} = \zeta^{\mathtt{k}_{\mathtt{h}i,j}}$ such that $\mathfrak{b} = g_2^{\mathtt{k}_{\mathtt{h}i,j}}$. To build a non-DDH triple $(\zeta, \mathfrak{a}, \mathfrak{b})$ we choose random α and set $\mathfrak{a} = \mathtt{k}_{\mathtt{p}3,i,j} = \zeta^{\mathtt{k}_{\mathtt{h}i,j}}$ and $\mathfrak{b} = g_2^{\alpha_j}$. To guarantee correctness we have to choose α such that $\alpha_j = \mathtt{k}_{\mathtt{h}j,i} + \beta_j$ for $\beta \in \ker \begin{pmatrix} g_{1,1} & 1 & g_{1,2} & h & c \\ 1 & g_{1,1} & 1 & 1 & d \end{pmatrix}$ for $j \in [1, 5]$. Note that this is possible because we know τ, which contains the secret Cramer-Shoup key. If we can build a distinguisher on h_0, we can now decide whether $(\zeta, \mathfrak{a}, \mathfrak{b})$ is a valid DDH triple or not.

$\mathcal{G}_4$ **:** In this game we choose $\mathtt{sk} \in_R \mathbb{G}_T$ at random in case we choose h_0 at random (the setting described in $\mathcal{G}_3$) and computation of $\mathtt{sk}$ on S_1 is successful. Since h_0 on S_1 is uniformly at random already and $\mathtt{sk} = h_0 h_x$, $\mathtt{view}_4$ is perfectly indistinguishable from $\mathtt{view}_3$.

$\mathcal{G}_5$ **:** Receiving an adversarially generated or modified C_1 or C_2 on behalf of client C, challenger $\mathcal{C}$ chooses $h_x \in_R \mathbb{G}_T$ uniformly at random instead of computing it with $\mathtt{Hash}_x$ if C_1 or C_2 do not encrypt the correct password share pw_1 or pw_2 respectively. We claim that $\mathtt{view}_5$ is computationally indistinguishable from $\mathtt{view}_4$. In this case we have $(C_0, C_1, C_1) \notin L_{\widehat{\mathrm{pw}}}$ with overwhelming probability. The claim therefore follows by a similar argument as in Game 3, i.e. from the DDH assumption in $\mathbb{G}_2$.

$\mathcal{G}_6$ **:** In this game we choose $\mathtt{sk} \in_R \mathbb{G}_T$ at random in case we choose h_x at random (the setting described in $\mathcal{G}_5$) and computation of $\mathtt{sk}$ on C is successful (projection keys $\mathtt{k}_{\mathtt{p}_1}$ and $\mathtt{k}_{\mathtt{p}_2}$ are correct). Since h_x on C is uniformly at random already and $\mathtt{sk} = h_0 h_x$, $\mathtt{view}_6$ is perfectly indistinguishable from $\mathtt{view}_5$.

$\mathcal{G}_7$ **:** In this game we replace computation of hash values h_0 and h_x with a lookup table with index $(\mathtt{k}_{\mathtt{h}1}, \mathtt{k}_{\mathtt{h}2}, L_{\mathrm{pw},\mathrm{pw}_2,\mathrm{pw}_2}, C_0)$ for h_0 and $(\mathtt{k}_{\mathtt{h}0}, L_{\mathrm{pw},\mathrm{pw}_2,\mathrm{pw}_2}, C_1, C_2)$ for h_x. If no such value exists, it is computed with the appropriate $\mathtt{Hash}$ or $\mathtt{PHash}$ function and stored in the lookup table. Due to the correctness of the used Cramer-Shoup TD-SPHF $\mathtt{view}_7$ is perfectly indistinguishable from $\mathtt{view}_6$.

$\mathcal{G}_8$ **:** Instead of computing $\mathtt{Hash}_0$ for S_1 in case pw' decrypted from C_0 is the same as pw, $\mathcal{C}$ draws a random $h_0 \in_R \mathbb{G}_T$. That is, in this game h_0 for S_1 is always chosen uniformly at random instead of computing it with $\mathtt{Hash}_0$. We claim that $\mathtt{view}_8$ is computationally indistinguishable from $\mathtt{view}_7$. The claim follows from the CCA-security of the labelled Cramer-Shoup encryption and the same argument as in Game 3,

i.e. from SXDH. In particular, we define $\mathcal{G}_7'$ and $\mathcal{G}_7''$ with computationally indistinguishable views from $\mathcal{G}_7$ as intermediate games before $\mathcal{G}_8$ such that the claim follows. Note that the following games modify the experiment only in the previously defined case. In $\mathcal{G}_7'$ challenger $\mathcal{C}$ computes C_1 for S_1 on a random value $\mathrm{pw}_1' \in_R \mathbb{Z}_q$, $\mathrm{pw}_1' \neq \mathrm{pw}_1$. The CCA-security of the encryption scheme ensure that $\mathtt{view}_{7'}$ is computationally indistinguishable from $\mathtt{view}_7$. In $\mathcal{G}_7''$ we choose a random $h_0 \in_R \mathbb{G}_T$ instead of using the distributed $\mathtt{Hash}_0$ computation (the protocol is still performed but the values are not used). Using the same argument as in $\mathcal{G}_3$, $\mathtt{view}_{7''}$ is computationally indistinguishable from $\mathtt{view}_{7'}$. The only difference between $\mathcal{G}_7''$ and $\mathcal{G}_8$ now is that $\mathcal{C}$ encrypts a random value instead of pw_1 in C_1 in $\mathcal{G}_7''$. The claim now follows by observing again that $\mathtt{view}_{7''}$ and $\mathtt{view}_8$ are computationally indistinguishable considering the CCA-security of the labelled Cramer-Shoup encryption scheme.

$\mathcal{G}_9$: In this game we choose $\mathtt{sk} \in_R \mathbb{G}_T$ at random in case we choose h_0 at random (the setting described in $\mathcal{G}_8$) and computation of $\mathtt{sk}$ on S_1 is successful. Since h_0 on S_1 is uniformly at random and $\mathtt{sk} = h_0 h_x$, $\mathtt{view}_9$ is perfectly indistinguishable from $\mathtt{view}_8$.

$\mathcal{G}_{10}$: Receiving correct C_1 or C_2, i.e. encrypting pw_1 and pw_2 respectively, on behalf of client C, challenger $\mathcal{C}$ chooses $h_x \in_R \mathbb{G}_T$ uniformly at random instead of computing it with $\mathtt{Hash}_x$. We claim that $\mathtt{view}_{10}$ is computationally indistinguishable from $\mathtt{view}_9$. Since we have $(C_0, C_1, C_1) \in L_{\widehat{\mathrm{pw}}}$ in this case, the claim follows by a similar argument as in Game 8, i.e. from the SXDH assumption.

$\mathcal{G}_{11}$: In this game we choose $\mathtt{sk} \in_R \mathbb{G}_T$ at random in case we choose h_0 at random (the setting described in $\mathcal{G}_{10}$) and computation of $\mathtt{sk}$ on C is successful (projection keys $\mathtt{k}_{\mathtt{p}_1}$ and $\mathtt{k}_{\mathtt{p}_2}$ are correct). Since h_x on C is uniformly at random already and $\mathtt{sk} = h_0 h_x$, $\mathtt{view}_{11}$ is perfectly indistinguishable from $\mathtt{view}_{10}$.

$\mathcal{G}_{12}$: The entire $\mathtt{crs}$ including ζ is chosen by challenger $\mathcal{C}$ in this experiment. The $\mathtt{view}_{12}$ is perfectly indistinguishable from $\mathtt{view}_{11}$ since this does not change anything else.

$\mathcal{G}_{13}$: Upon receiving C_1 and C_2, encrypting correct password shares, $\mathcal{C}$ uses $\mathtt{THash}_0$ to compute h_0 on client C instead of $\mathtt{PHash}_0$. This is possible because $\mathcal{C}$ now knows trapdoor τ'. Due to TD-SPHF soundness, $\mathtt{view}_{13}$ is perfectly indistinguishable from $\mathtt{view}_{12}$.

$\mathcal{G}_{14}$: Upon receiving C_0, encrypting correct password, $\mathcal{C}$ uses $\mathtt{THash}_x$ to compute h_x on server S_1 instead of $\mathtt{PHash}_x$. This is again possible because $\mathcal{C}$ now knows trapdoor τ'. Due to TD-SPHF soundness, $\mathtt{view}_{14}$ is perfectly indistinguishable from $\mathtt{view}_{13}$.

$\mathcal{G}_{15}$: Instead of encrypting the correct password pw in C_0 on behalf of client C, $\mathcal{C}$ encrypts 0 (which is not a valid password). We claim that $\mathtt{view}_{15}$ is computationally indistinguishable from $\mathtt{view}_{14}$ under the DDH assumption in $\mathbb{G}_1$, i.e. the CCA-security of the Cramer-Shoup encryption. Note that encryption randomness r is not used in the computation of h_0 anymore such that the claim follows from the Cramer-Shoup CCA-security.

$\mathcal{G}_{16}$: Instead of encrypting the correct password share pw_i in C_i on behalf of server S_i with $i \in [1, 2]$, $\mathcal{C}$ encrypts a random element $\mathrm{pw}_i' \in_R \mathbb{Z}_q$. We claim that $\mathtt{view}_{16}$ is computationally indistinguishable from $\mathtt{view}_{15}$ under the DDH assumption in $\mathbb{G}_1$, i.e. the CCA-security of the Cramer-Shoup encryption. Note that the probability for $\mathrm{pw}_i' = \mathrm{pw}_i$ is negligible such that the claim follows from the Cramer-Shoup CCA-security.

$\mathcal{G}_{17}$: Instead of the challenger $\mathcal{C}$ simulating the protocol execution the ideal functionality $\mathcal{F}_{\mathrm{2PAKE}}$ is used to interact with the ideal-world adversary $\mathtt{SIM}$. While this game is structurally different from $\mathcal{G}_{16}$ their executions are indistinguishable. This combined with the following description of the ideal world adversary $\mathtt{SIM}$ concludes the proof.

Simulator We now describe the simulator $\texttt{SIM}$ that is used in the last experiment and acts as an attacker in the ideal world against the ideal functionality $\mathcal{F}_{\text{2PAKE}}$, interacting with the real world adversary $\mathcal{A}$. It uses a real-world adversary $\mathcal{A}$ in a way that the environment $\mathcal{Z}$ cannot distinguish whether it is interacting with $\mathcal{A}$ and honest protocol participants in the real world, or with $\texttt{SIM}$ and dummy protocol participants (simulated by $\mathcal{F}_{\text{2PAKE}}$) in the ideal world. We describe $\texttt{SIM}$ for a single session $\texttt{sid} = (C, S_1, S_2)$. The security then follows from the UC composition theorem [16], covering multiple sessions of the protocol, and joint-state UC composition theorem [21], covering the fact that $\mathcal{F}_{\texttt{CA}}$ and $\mathcal{F}_{\texttt{crs}}$ create a joint state for all sessions and participants. As before, we assume that 0 is not a valid password.

First, $\texttt{SIM}$ generates $\texttt{crs} = (q, g_{1,1}, g_{1,2}, h, c, d, \mathbb{G}_1, g_2, \zeta, \mathbb{G}_2, \mathbb{G}_T, e, H_k)$ with Cramer-Shoup secret key as trapdoor $\tau = (x_1, x_2, y_1, y_2, z)$ and second trapdoor τ' for $\zeta = g_2^{\tau'}$ to answer all $\mathcal{F}_{\texttt{crs}}$ queries with $\texttt{crs}$. Further, $\texttt{SIM}$ generates ElGamal key pairs (g^{z_1}, z_1) and (g^{z_2}, z_2), and responds to $\texttt{Retrieve}(S_i)$ queries to $\mathcal{F}_{\texttt{CA}}$ from S_i with $(\texttt{Retrieve}, S_i, (g^{z_i}, z_i))$ for $i \in \{1, 2\}$ and with $(\texttt{Retrieve}, S_i, g^{z_i})$ to all other request. We describe different scenarios in which the simulator operates. First we describe simulation of the initial KEXInit call before showing the way $\texttt{SIM}$ handles different input messages and the key generation. The simulator essentially has to ensure that the functionality chooses random, correct session keys if the execution is correct and random, independent ones in case of an error during the execution.

When receiving $(\texttt{KEX}, \texttt{sid}, \texttt{qid}, P)$ with $\texttt{sid} = (C, S_1, S_2)$ and $P \in \{C, S_1, S_2\}$ from $\mathcal{F}_{\text{2PAKE}}$, $\texttt{SIM}$ starts simulation of the protocol for protocol participant P by computing ciphertext, projection key pair $M_i = (C_i, \texttt{k}_{\texttt{p}_i})$ for $i \in \{0, 1, 2\}$, encrypting a dummy value (0 for $P = C$ and a random value $\alpha'_i \in_R \mathbb{Z}_q$ for $P = S_i$, $i \in \{1, 2\}$). $\texttt{SIM}$ outputs the computed $(C_i, \texttt{k}_{\texttt{p}_i})$ to $\mathcal{A}$. The first round of messages is handled as follows.

i) When any party receives an adversarially generated but well formed first message M_i, $i \in \{1, 2\}$ from uncorrupted S_i, i.e. $\texttt{VerKp}$ on the projection key $\texttt{k}_{\texttt{p}_i}$ is 1, $\texttt{SIM}$ queries $(\texttt{FA}, \texttt{sid}, \texttt{qid})$, which marks the session $\texttt{failed}$ for the receiving party and thus ensures that the party receives an independent, random session key (if any) on a NewKey query.

ii) When any party receives an adversarially generated but well formed first message M_2 from a corrupted S_2 while S_1 is not corrupted, $\texttt{SIM}$ decrypts C_2 to α'_2. If this value is not correct, $\alpha'_2 \neq \alpha_2$ (the party is corrupted such that $\texttt{SIM}$ knows the correct value), $\texttt{SIM}$ queries $(\texttt{FA}, \texttt{sid}, \texttt{qid})$ to ensure independent session keys on NewKey queries.

iii) When client C receives an adversarially generated but well formed first message M_1 from a corrupted S_1 while S_2 is not corrupted, $\texttt{SIM}$ decrypts C_1 to α'_1. If this value is *not* correct, $\alpha'_1 \neq \alpha_1$, $\texttt{SIM}$ queries $(\texttt{FA}, \texttt{sid}, \texttt{qid})$ to ensure independent session keys on NewKey queries.

iv) When any party receives adversarially generated but well formed first messages M_1, M_2 from corrupted S_1, S_2, $\texttt{SIM}$ decrypts C_1 and C_2 to α'_1, α'_2 respectively, and verifies their correctness against α_1 and α_2. If they are correct, $\texttt{SIM}$ computes $h_0 \leftarrow \texttt{THash}_0(\texttt{k}_{\texttt{p}_1}, \texttt{k}_{\texttt{p}_2}, L_{\text{pw},\text{pw}_1,\text{pw}_2}, C_0, \tau')$, $h_x \leftarrow \texttt{Hash}_x(\texttt{k}_{\texttt{p}_0}, L_{\widehat{\text{pw}}}, C_1, C_2)$, and $\texttt{sk}_C = h_0 \cdot h_x$. Otherwise choose a random $\texttt{sk}_C \in \mathbb{G}_T$.

v) When an honest S_1 or S_2 receives an adversarially generated but well formed first message M_0, i.e. $\texttt{VerKp}$ on $\texttt{k}_{\texttt{p}_0}$ is $\texttt{true}$, $\texttt{SIM}$ extracts pw′ from C_0 and sends $(\texttt{TP}, \texttt{sid}, \texttt{qid}, C, \text{pw}')$ to $\mathcal{F}_{\text{2PAKE}}$. If the functionality replies with "correct guess", $\texttt{SIM}$ uses pw′, $\texttt{crs}$ and τ' to compute $h_x \leftarrow \texttt{THash}_x(\texttt{k}_{\texttt{p}_0}, L_{\widehat{\text{pw}}}, C_1, C_2, \tau')$, $h_0 \leftarrow \texttt{Hash}_0(\texttt{k}_{\texttt{h}1}, \texttt{k}_{\texttt{h}2}, L_{\text{pw},\text{pw}_1,\text{pw}_2}, C_0)$, and $\texttt{sk}_S = h_0 \cdot h_x$.

vi) If verification of any $\texttt{k}_{\texttt{p}_i}$ fails at a recipient, $\texttt{SIM}$ aborts the session for the receiving participant.

If a party does not abort, it proceeds as follows. After C received all ciphertext, projection key pair messages and the previously described checks were performed $\texttt{SIM}$ sends $(\texttt{NK}, \texttt{sid}, \texttt{qid}, C, \texttt{sk}_C)$ to $\mathcal{F}_{\text{2PAKE}}$ if an $\texttt{sk}_C$ for this session exists, or $(\texttt{NK}, \texttt{sid}, \texttt{qid}, C, \perp)$ otherwise. After S_1 and S_2 received all ciphertext, projection key pair messages and the previously described checks were performed, $\texttt{SIM}$ simulates all further messages for honest parties, i.e. $\texttt{PHash}_x$ and $\texttt{Hash}_0$ computation between S_1 and S_2, with random elements and simulated zero-knowledge proofs. If all messages received by S_1 are oracle generated, send $(\texttt{NK}, \texttt{sid}, \texttt{qid}, S_1, \texttt{sk}_S)$ to $\mathcal{F}_{\text{2PAKE}}$ if this session is $\texttt{compromised}$ and $(\texttt{NK}, \texttt{sid}, \texttt{qid}, S_1, \perp)$ if not. If any $\texttt{PHash}_x$ or $\texttt{Hash}_0$ message received by S_1 can not be verified, i.e. validation of the zero-knowledge proof fails, $\texttt{SIM}$ does nothing and aborts the session for S_1.

5 Conclusion

This paper proposed the first UC-secure 2PAKE and introduced Trapdoor Distributed Smooth Projective Hashing (TD-SPHF) as its building block. The proposed 2PAKE protocol uses a common reference string and the SXDH assumption on bilinear groups and is efficient thanks to the simulatability of TD-SPHF.

References

1. M. Abdalla, F. Benhamouda, O. Blazy, C. Chevalier, and D. Pointcheval. SPHF-Friendly Non-interactive Commitments. In *ASIACRYPT (1)*, volume 8269 of *Lecture Notes in Computer Science*, pages 214–234. Springer-Verlag, 2013. 10
2. M. Abdalla, F. Benhamouda, and D. Pointcheval. Removing Erasures with Explainable Hash Proof Systems. Cryptology ePrint Archive, Report 2014/125, 2014. UC, adaptive. 10
3. M. Abdalla, C. Chevalier, and D. Pointcheval. Smooth Projective Hashing for Conditionally Extractable Commitments. In *CRYPTO'09*, volume 5677 of *Lecture Notes in Computer Science*, pages 671–689. Springer-Verlag, 2009. UC, adaptive. 10
4. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In *Proceedings of the 8th international conference on Theory and Practice in Public Key Cryptography*, PKC'05, pages 65–84, Berlin, Heidelberg, 2005. Springer-Verlag. 1
5. G. Ateniese, J. Camenisch, S. Hohenberger, and B. de Medeiros. Practical group signatures without random oracles. *IACR Cryptology ePrint Archive*, 2005:385, 2005. 2
6. L. Ballard, M. Green, B. de Medeiros, and F. Monrose. Correlation-resistant storage via keyword-searchable encryption. *IACR Cryptology ePrint Archive*, 2005:417, 2005. 2
7. B. Barak, Y. Lindell, and T. Rabin. Protocol Initialization for the Framework of Universal Composability. *IACR Cryptology ePrint Archive*, 2004:6, 2004. 5
8. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 139–155, Berlin, Heidelberg, 2000. Springer-Verlag. 1
9. S. M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise. In *ACM CCS'93*, pages 244–250. ACM, 1993. 1
10. F. Benhamouda, O. Blazy, C. Chevalier, D. Pointcheval, and D. Vergnaud. New Techniques for SPHFs and Efficient One-Round PAKE Protocols. In *CRYPTO'13*, volume 8042 of *Lecture Notes in Computer Science*, pages 449–475. Springer-Verlag, 2013. 2, 3, 4, 5, 6, 17, 18
11. F. Benhamouda and D. Pointcheval. Trapdoor smooth projective hash functions. *eprint.iacr.org*, 2013. 12
12. F. Benhamouda and D. Pointcheval. Verifier-based password-authenticated key exchange: New models and constructions. *IACR Cryptology ePrint Archive*, 2013:833, 2013. 1
13. J. Brainard and A. Juels. A new two-server approach for authentication with short secrets. *USENIX03*, 2003. 1
14. J. Camenisch, R. R. Enderlein, and G. Neven. Two-Server Password-Authenticated Secret Sharing UC-Secure Against Transient Corruptions. *IACR Cryptology ePrint Archive*, 2015:006, 2015. 2
15. J. Camenisch, A. Lysyanskaya, and G. Neven. *Practical yet universally composable two-server password-authenticated secret sharing*, page 525536. ACM, 2012. 2
16. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, FOCS'01, page 136, Washington, DC, USA, 2001. IEEE Computer Society. 14
17. R. Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219. IEEE Computer Society, 2004. 5
18. R. Canetti and M. Fischlin. Universally Composable Commitments. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 2001. 5
19. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally Composable Password-Based Key Exchange. In *Proceedings of the 24th annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, pages 404–421, Berlin, Heidelberg, 2005. Springer-Verlag. 1, 10
20. R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *EUROCRYPT'01*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001. 10

21. R. Canetti and T. Rabin. Universal Composition with Joint State. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer-Verlag, 2003. 14
22. I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT'00*, volume 1807 of *Lecture Notes in Computer Science*, pages 418–430. Springer, 2000. 3
23. R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. *ACM Trans. Inf. Syst. Secur.*, 9(2):181–234, may 2006. 16
24. C. Gentry, P. D. MacKenzie, and Z. Ramzan. A Method for Making Password-Based Key Exchange Resilient to Server Compromise. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference*, volume 4117 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2006. 1
25. hashcat. hashcat - advanced password recovery. `http://hashcat.net/`, 2014. Accessed: 1/12/2014. 1
26. S. Jarecki, A. Kiayias, and H. Krawczyk. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. In *ASIACRYPT'14*, volume 8874 of *Lecture Notes in Computer Science*, pages 233–253. Springer-Verlag, 2014. 2
27. S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *EUROCRYPT'00*, volume 1807 of *Lecture Notes in Computer Science*, pages 221–242. Springer, 2000. 3
28. H. Jin, D. Wong, and Y. Xu. An efficient password-only two-server authenticated key exchange system. *Information and Communications Security*, page 4456, 2007. 1
29. J. Katz, P. MacKenzie, G. Taban, and V. Gligor. Two-server password-only authenticated key exchange. In *ACNS'05*, volume 3531 of *Lecture Notes in Computer Science*, pages 1–16, 2005. 2pake. 1, 2, 4, 21, 22
30. J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. In *Proceedings of the 8th conference on Theory of cryptography*, TCC'11, pages 293–310, Berlin, Heidelberg, 2011. Springer-Verlag. 3, 16, 17
31. F. Kiefer and M. Manulis. Distributed Smooth Projective Hashing and its Application to Two-Server Password Authenticated Key Exchange. In *ACNS'14*, volume 2020 of *Lecture Notes in Computer Science*, pages 344–360. Springer-Verlag, 2014. 1, 2, 4, 5, 6, 9, 10
32. P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. *Advances in CryptologyCRYPTO 2002*, page 141, 2002. 2
33. A. Miyaji, M. Nakabayashi, and S. Takano. Characterization of elliptic curve traces under fr-reduction. In *ICISC'00*, volume 2015 of *LNCS*, pages 90–108. Springer, 2000. 2
34. Openwall. John the Ripper password cracker. `http://www.openwall.com/john/`, 2014. Accessed: 1/12/2014. 1
35. T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1991. 2
36. M. D. Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. *Advances in CryptologyEUROCRYPT 2003*, page 507523, 2003. 2
37. M. Szydlo and B. S. K. Jr. Proofs for Two-Server Password Authentication. In *CT-RSA'05*, volume 3376 of *Lecture Notes in Computer Science*, pages 227–244. Springer-Verlag, 2005. 1, 21
38. T. Wu. RFC 2945 - The SRP Authentication and Key Exchange System, sep 2000. 1
39. Y. Yang, R. Deng, and F. Bao. A practical password-based two-server authentication and key exchange system. *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING*, 3(2):105114, 2006. 1

A Smooth Projective Hashing

In a nutshell, smoothness ensures that the hash value always looks random in $\mathbb{G}$ when computed on an element not in the language, while pseudorandomness ensures that it looks random in $\mathbb{G}$ when computed on an element in the language. Note again that we are only concerned with KV-SPHF that have word-independent keys and offer adaptive smoothness (first proposed in [30]). The corresponding notion of adaptive smoothness with word-independent keys is defined as follows. For any function $f : \mathbb{G} \mapsto \mathcal{C} \setminus L_{\text{pw}}$ the following distributions are statistically ε-close:

$$\begin{aligned} &\{(\mathtt{k_p}, h) \mid \mathtt{k_h} \xleftarrow{R} \mathtt{KGen_H}(L_{\text{pw}}); \mathtt{k_p} \leftarrow \mathtt{KGen_P}(\mathtt{k_h}, L_{\text{pw}}); h \leftarrow \mathtt{Hash}(\mathtt{k_h}, L_{\text{pw}}, f(\mathtt{k_p}))\} \\ \stackrel{\varepsilon}{=} &\{(\mathtt{k_p}, h) \mid \mathtt{k_h} \xleftarrow{R} \mathtt{KGen_H}(L_{\text{pw}}); \mathtt{k_p} \leftarrow \mathtt{KGen_P}(\mathtt{k_h}, L_{\text{pw}}); h \in_R \mathbb{G}\} \end{aligned}$$

Gennaro and Lindell [23] introduced pseudorandomness of SPHFs to show that `Hash` and `PHash` are the only way to compute the hash value even though the adversary knows some tuples $(\mathtt{k_p}, C, \mathtt{Hash}(\mathtt{k_h}, L_{\text{pw}}, C))$ for

$C \in L_{\text{pw}}$. An SPHF is pseudorandom if the hash values produced by $\texttt{Hash}$ and $\texttt{PHash}$ are indistinguishable from random without the knowledge of the uniformly chosen hash key $\texttt{k}_\texttt{h}$ or a witness w, i.e. for all $C \in L_{\text{pw}}$ the following distributions are computationally ε-close:

$$\begin{aligned}&\{(\texttt{k}_\texttt{p}, C, h) \mid \texttt{k}_\texttt{h} \xleftarrow{R} \texttt{KGen}_\texttt{H}(L_{\text{pw}}); \texttt{k}_\texttt{p} \leftarrow \texttt{KGen}_\texttt{P}(\texttt{k}_\texttt{h}, L_{\text{pw}}); h \leftarrow \texttt{Hash}(\texttt{k}_\texttt{h}, L_{\text{pw}}, C)\}\\ \overset{\varepsilon}{=} &\{(\texttt{k}_\texttt{p}, C, h) \mid \texttt{k}_\texttt{h} \xleftarrow{R} \texttt{KGen}_\texttt{H}(L_{\text{pw}}); \texttt{k}_\texttt{p} \leftarrow \texttt{KGen}_\texttt{P}(\texttt{k}_\texttt{h}, L_{\text{pw}}); h \in_R \mathbb{G}\}\end{aligned}$$

The property of pseudorandomness from [30] is expected to hold even if hashing keys and ciphertexts are re-used.

Definition 7 (Pseudorandomness). *An* SPHF Π *offers pseudorandomness if for all PPT algorithms* $\mathcal{A}$ *and polynomials* l *there exists a negligible function* $\varepsilon(\cdot)$ *such that*

$$\mathsf{Adv}^{\text{Pr}}_{\Pi,\mathcal{A}} = \left|\Pr[\mathsf{Exp}^{\text{Pr}}_{\Pi,\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \varepsilon(\lambda)$$

$\mathsf{Exp}^{\text{Pr}}_{\Pi,\mathcal{A}}(\lambda)$: *Choose* $b \in_R \{0,1\}$*, call* $b' \leftarrow \mathcal{A}^{\Omega^{\mathcal{L}}_{\texttt{pk}}(\cdot), \texttt{Dec}^{\mathcal{L}}_{\pi}(\cdot)}(\lambda, \texttt{k}_{\texttt{p}1}, \ldots, \texttt{k}_{\texttt{p}l})$ *with* $\texttt{k}_{\texttt{p}i} \leftarrow \texttt{KGen}_\texttt{P}(\texttt{k}_{\texttt{h}i}, L_{\text{pw}}, C)$ *and* $\texttt{k}_{\texttt{h}i} \leftarrow \texttt{KGen}_\texttt{H}(L_{\text{pw}})$ *for all* $i \in 1, \ldots, l$*. Return* $b = b'$*.*

$\Omega^{\mathcal{L}}_{\texttt{pk}}(\ell, \text{pw})$ *returns elements* $C \in L_{\texttt{aux}}$ *with* $C \leftarrow \texttt{Enc}^{\mathcal{L}}_{\texttt{pk}}(\ell, \text{pw}; r)$ *using encryption algorithm* $\mathcal{L}$ *and label* ℓ*. It additionally returns* $\texttt{Hash}(\texttt{k}_{\texttt{h}i}, L_{\text{pw}}, C)$ *if* $b = 0$ *or* $h_i \in_R \mathbb{G}$ *if* $b = 1$ *for all* $i \in 1, \ldots, l$*.*
$\texttt{Dec}^{\mathcal{L}}_{\pi}(\ell, C)$ *decrypts the ciphertext* C *with label* ℓ *if* (ℓ, C) *was not obtained from* $\Omega^{\mathcal{L}}_{\texttt{pk}}$*.*

A.1 SPHF on Cramer-Shoup Ciphertexts

Benhamouda et al. propose a new perfectly smooth SPHF for labelled Cramer-Shoup encryptions in [10]. The SPHF is defined as follows:

- $\texttt{KGen}_\texttt{H}(L_{\text{pw}})$ return $\texttt{k}_\texttt{h} = (\eta_1, \eta_2, \theta, \mu, \nu) \in_R \mathbb{Z}_q^{1\times 5}$
- $\texttt{KGen}_\texttt{P}(\texttt{k}_\texttt{h}, L_{\text{pw}})$ returns $\texttt{k}_\texttt{p} = (\texttt{k}_{\texttt{p}1} = g_1^{\eta_1} g_2^{\theta} h^{\mu} c^{\nu}, \texttt{k}_{\texttt{p}2} = g_1^{\eta_2} d^{\nu})$
- $\texttt{Hash}(\texttt{k}_\texttt{h}, L_{\text{pw}}, C)$ computes $h = u_1^{\eta_1 + \xi\eta_2} u_2^{\theta} (e/g_1^{\text{pw}})^{\mu} v^{\nu}$
- $\texttt{PHash}(\texttt{k}_\texttt{p}, L_{\text{pw}}, C, r)$ computes $h = (\texttt{k}_{\texttt{p}1} \texttt{k}_{\texttt{p}2}^{\xi})^r$

B Trapdoor Smooth Projective Hashing

Correctness of T-SPHFs extends correctness of SPHFs by the statement that for every valid ciphertext C, generated by $\mathcal{L}$, and honestly generated keys $\texttt{k}_\texttt{h}$ and $\texttt{k}_\texttt{p}$, it holds that $\texttt{VerKp}(\texttt{k}_\texttt{p}, L_{\text{pw}}) = 1$ and $\texttt{Hash}(\texttt{k}_\texttt{h}, L_{\text{pw}}, C) = \texttt{THash}(\texttt{k}_\texttt{p}, L_{\text{pw}}, C, \tau')$. To capture soundness of T-SPHFs [10] introduces (t, ε)-*soundness*, complementing the previous correctness extension.

Definition 8 ((t, ε)-soundness). *Given* $\texttt{crs}$*,* $\texttt{crs}'$ *and* τ*, no adversary running in time at most* t *can produce a projection key* $\texttt{k}_\texttt{p}$*, a password* pw*, a word* C*, and valid witness* r *such that* $\texttt{k}_\texttt{p}$ *is valid, i.e.* $\texttt{VerKp}(\texttt{k}_\texttt{p}, L_{\text{pw}}) = 1$*, but* $\texttt{THash}(\texttt{k}_\texttt{p}, L_{\text{pw}}, C, \tau') \neq \texttt{PHash}(\texttt{k}_\texttt{p}, L_{\text{pw}}, C, r)$ *with probability at least* $\varepsilon(\lambda)$*. Perfect soundness states that the property holds for any* t *and any* $\varepsilon(\lambda) > 0$*.*

As statistical smoothness is impossible for T-SPHF, [10] introduces the notion computational smoothness, which is similar to the definition of pseudorandomness for SPHFs.

Definition 9 (Computational Smoothness [10]). *An SPHF is* (t, ε)*-smooth if for all adversaries* $\mathcal{A}$ *running in time at most* t

$$\mathsf{Adv}^{\texttt{smooth}-\texttt{b}}_{\Pi,\mathcal{A}} = \left|\Pr[\mathsf{Exp}^{\texttt{smooth}-1}_{\text{SPHF},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\texttt{smooth}-0}_{\text{SPHF},\mathcal{A}}(\lambda) = 1]\right| \leq \varepsilon(\lambda).$$

$\mathsf{Exp}^{\mathtt{smooth-b}}_{\mathrm{SPHF},\mathcal{A}}(\lambda)$: *Generate* $(\mathtt{crs}', \tau') \xleftarrow{R} \mathtt{TSetup}(\mathtt{crs})$. *The adversary, given* $\mathtt{crs}, \mathtt{crs}'$ *and* τ*, is then allowed to query* $\mathcal{O}_{\mathtt{KGen_P}}(\cdot)$ *and* $\mathcal{O}_{\mathtt{Hash}_b}(\cdot)$ *once before returning a bit* b'*. Return* $b = b'$.

On input pw *the* $\mathcal{O}_{\mathtt{KGen_P}}$ *oracle draws a new hash key* $\mathtt{k_h}$ *for T-SPHF on* L_{pw}*, computes the according projection key* $\mathtt{k_p}$ *using* $\mathtt{KGen_P}$*, and returns it to the adversary. The* $\mathcal{O}_{\mathtt{Hash}}$ *oracle returns* $h \leftarrow \mathtt{Hash}$ *honestly computed on input ciphertext* C *if* $b = 0$ *or* $C \in L_{\mathrm{pw}}$*, and* $h \in_R \mathbb{G}$ *if* $b = 1$.

B.1 T-SPHF on Cramer-Shoup Ciphertexts

Benhamouda et al. propose a T-SPHF for labelled CS ciphertexts in [10] under the SXDH assumption. The T-SPHF is a straight-forward extension of the previously described SPHF in labelled CS ciphertexts. Let $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ denote a bilinear group and replace $\mathbb{G}$ from the previous SPHF with $\mathbb{G}_1$ and g_1, g_2 by $g_{1,1}, g_{1,2}$, generators for $\mathbb{G}_1$. All other previous parameters are in $\mathbb{G}_1$ instead of $\mathbb{G}$ and g_2 is generator of $\mathbb{G}_2$. The additional algorithms for T-SPHF and changes to the hash functions are defined as follows.

- $\mathtt{TSetup}(\mathtt{crs})$ draws a random $\tau' \in_R \mathbb{Z}_q$ and sets $\mathtt{crs}' = \zeta = g_2^{\tau'}$.
- $\mathtt{KGen_P}(\mathtt{k_h}, L_{\mathrm{pw}})$ generates $\mathtt{k_p} = (\mathtt{k_{p_1}} = g_1^{\eta_1} g_2^{\theta} h^{\mu} c^{\nu}, \mathtt{k_{p_2}} = g_1^{\eta_2} d^{\nu}, \mathtt{k_{p_3}})$ with $\mathtt{k_{p_3}} = (\chi_{1,1}, \chi_{1,2}, \chi_2, \chi_3, \chi_4)$ for $\chi_{1,1} = \zeta^{\eta_1}, \chi_{1,2} = \zeta^{\eta_2}, \chi_2 = \zeta^{\theta}, \chi_3 = \zeta^{\mu}, \chi_4 = \zeta^{\nu}$
- $\mathtt{Hash}(\mathtt{k_h}, L_{\mathrm{pw}}, C)$ computes $h' = u_1^{\eta_1 + \xi\eta_2} u_2^{\theta} (e/g_1^{\mathrm{pw}})^{\mu} v^{\nu}$ as before and outputs $h = e(h', g_2)$
- $\mathtt{PHash}(\mathtt{k_p}, L_{\mathrm{pw}}, C, r)$ computes $h' = (\mathtt{k_{p_1}} \mathtt{k_{p_2}}^{\xi})^r$ as before and outputs $h = e(h', g_2)$
- $\mathtt{VerKp}(\mathtt{k_p}, L_{\mathrm{pw}})$ verifies that $e(\mathtt{k_{p_1}}, \mathtt{crs}') \stackrel{?}{=} e(g_{1,1}, \chi_{1,1}) \cdot e(g_{1,2}, \chi_2) \cdot e(h_1, \chi_3) \cdot e(c, \chi_4)$ and $e(\mathtt{k_{p_2}}, \mathtt{crs}') \stackrel{?}{=} e(g_{1,1}, \chi_{1,2}) \cdot e(d, \chi_4)$
- $\mathtt{THash}(\mathtt{k_p}, L_{\mathrm{pw}}, C, \tau')$ computes $\left[e(u_1, \chi_{1,1}\chi_{1,2}^{\xi}) \cdot e(u_2, \chi_2) \cdot e(e/g_{1,1}^{\mathrm{pw}}, \chi_3) \cdot e(v, \chi_4)\right]^{1/\tau'}$

C Distributed Smooth Projective Hashing

The idea of D-SPHF security is to combine smoothness and pseudorandomness in one security experiment where the attacker, with access to one server, tries to distinguish between real and random hash values. Let $\{(C_j, S_{k,1}, S_{l,2})\}_{C_j \in \mathcal{C}, S_{k,1}, S_{l,2} \in \mathcal{S}}$ denote all tuples $(C_j, S_{k,1}, S_{l,2})$ such that client $C_j \in \mathcal{C}$ knows pw and server $S_{k,1}, S_{l,2} \in \mathcal{S}$ each know according pw_1 and pw_2 respectively. We say C is *registered* with (S_1, S_2). The additional indices j, k, l denote the instance of the respective participant. Parties without specified role are denoted P_a and P_b.

Definition 10 (D-SPHF Security). *A D-SPHF protocol* Π *is secure if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\varepsilon(\cdot)$ *such that :*

$$\mathsf{Adv}^{SPHF^{\varkappa}}_{\Pi,\mathcal{A}}(\lambda) = \left| \Pr[\mathsf{Exp}^{SPHF^{\varkappa}}_{\Pi,\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \right| \leq \varepsilon(\lambda)$$

$\mathsf{Exp}^{SPHF^{\varkappa}}_{\Pi,\mathcal{A}}(\lambda)$: *Choose* $b \in_R \{0,1\}$*, call* $b' \leftarrow \mathcal{A}^{\mathtt{Setup}(\cdot),\mathsf{Send}(\cdot),\mathsf{Test}(\cdot)}(\lambda, \mathrm{pw}_2, \mathcal{L}, \mathtt{crs})$ *and return* $b = b'$.

- $\mathtt{Setup}(C, S_1, S_2)$ *initialises new instances with* (pw, S_1, S_2) *for* C *registered with* (S_1, S_2)*, i.e.* $(\mathrm{pw}_1, S_1, C, S_2)$ *for* S_1 *and* $(\mathrm{pw}_2, S_2, C, S_1)$ *for* S_2*, and returns* $((\mathtt{k_{p_0}}, C_0), (\mathtt{k_{p_1}}, C_1))$ *with* $C_0 \leftarrow \mathtt{Enc}^{\mathcal{L}}_{\mathtt{pk}}(\ell, \mathrm{pw}; r_0)$*,* $C_1 \leftarrow \mathtt{Enc}^{\mathcal{L}}_{\mathtt{pk}}(\ell, \mathrm{pw}_1; r_1)$ *and* $\mathtt{k_{p_i}} \leftarrow \Pi.\mathtt{KGen_P}(\mathtt{k_{h}}i, L_{\widehat{\mathrm{pw}}})$ *for* $\mathtt{k_{h}}i \leftarrow \Pi.\mathtt{KGen_H}(L_{\widehat{\mathrm{pw}}})$.
- $\mathsf{Send}(P_a, P_b, m)$ *sends message* m *with alleged originator* P_b *to* P_a *and returns* P_a*'s resulting message* m' *if any.*
- $\mathsf{Test}(P_{i,j})$ *returns two hash values* (h_0, h_x) *if* $P_{i,j}$ *is from* $\mathcal{C}$ *or plays the role of* S_1*. If the global bit* b *is* 0*, the hash values are chosen uniformly at random, otherwise the hash values are computed according to protocol specification* Π.

C.1 Cramer-Shoup D-SPHF

As before for SPHF and T-SPHF we give an instantiation of D-SPHF over the language of Cramer-Shoup ciphertexts C. The ciphertexts are created as $C_i = (u_{1,i}, u_{2,i}, e_i, v_i) \leftarrow \mathtt{Enc}^{\mathrm{CS}}_{\mathtt{pk}}(\ell_i, \mathrm{pw}_i; r_i)$ for $i = 0, 1, 2$, $\mathrm{pw} = \mathrm{pw}_0$ and $\mathrm{pw} = \mathrm{pw}_1 + \mathrm{pw}_2$, where ℓ_i consists of participating parties and the party's projection key. We define modified decryption as $\mathtt{Dec}'(C) = e_0 \cdot u_{1,0}^{-z}$ and use the homomorphic property of u_1 and e of the CS ciphertext such that $\mathtt{Dec}'(C) = (e_1 \cdot e_2) \cdot (u_{1,1} \cdot u_{1,2})^{-z}$. The Cramer-Shoup D-SPHF can now be described as:

- $\mathtt{KGen}_{\mathtt{H}}(L_{\widehat{\mathrm{pw}}})$ returns $\mathtt{k}_{\mathtt{h}} = (\eta_1, \eta_2, \theta, \mu, \nu) \in_R \mathbb{Z}_p^{1\times 5}$
- $\mathtt{KGen}_{\mathtt{P}}(\mathtt{k}_{\mathtt{h}}, L_{\widehat{\mathrm{pw}}})$ returns $\mathtt{k}_{\mathtt{p}} = (\mathtt{k}_{\mathtt{p}1} = g_1^{\eta_1} g_2^{\theta} h^{\mu} c^{\nu}, \mathtt{k}_{\mathtt{p}2} = g_1^{\eta_2} d^{\nu})$
- $\mathtt{Hash}_x(\mathtt{k}_{\mathtt{h}0}, L_{\widehat{\mathrm{pw}}}, C_1, C_2)$ computes

$$h_x = (u_{1,1} \cdot u_{1,2})^{\eta_{1,0} + (\xi_1 + \xi_2)\eta_{2,0}} (u_{2,1} \cdot u_{2,2})^{\theta_0} ((e_1 \cdot e_2)/g_1^{\mathrm{pw}})^{\mu_0} (v_1 \cdot v_2)^{\nu_0}$$

- $\mathtt{PHash}_x(\mathtt{k}_{\mathtt{p}0}, L_{\widehat{\mathrm{pw}}}, C_1, C_2, r_1, r_2)$ computes

$$h_x = \mathtt{k}_{\mathtt{p}1,0}^{r_1 + r_2} \mathtt{k}_{\mathtt{p}2,0}^{\xi_1 r_1 + \xi_2 r_2}$$

- $\mathtt{Hash}_0(\mathtt{k}_{\mathtt{h}1}, \mathtt{k}_{\mathtt{h}2}, L_{\widehat{\mathrm{pw}}}, C_0)$ computes

$$h_0 = u_{1,0}^{\eta_{1,1} + \eta_{1,2} + \xi_0(\eta_{2,1} + \eta_{2,2})} u_{2,0}^{\theta_1 + \theta_2} (e_0/g_1^{\mathrm{pw}})^{\mu_1 + \mu_2} v^{\nu_1 + \mu_2}$$

- $\mathtt{PHash}_0(\mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2}, L_{\widehat{\mathrm{pw}}}, C_0, r_0)$ computes $h_0 = (\mathtt{k}_{\mathtt{p}1,1} \mathtt{k}_{\mathtt{p}1,2})^{r_0} (\mathtt{k}_{\mathtt{p}2,1} \mathtt{k}_{\mathtt{p}2,2})^{r_0 \xi_0}$

Distributed computation of D-SPHF is defined in the following $\mathtt{PHash}_x^D$ and $\mathtt{Hash}_0^D$ protocols. It uses ElGamal encryption to secure communication between the two servers. Let $C = (u, e) \leftarrow \mathtt{Enc}^{\mathrm{EG}}_{\mathtt{pk}}(m; r)$ with $u = g^r$ and $e = h^r g^m$ denote an El-Gamal ciphertext. Note that we assume $m \in \mathbb{Z}_q$ and $\mathbb{G}$ is a cyclic group of prime order q with generator g such that $g^m \in \mathbb{G}$. The ElGamal public key is defined as $\mathtt{pk} = (q, \mathbb{G}, g, h)$ with $h = g^z$ such that $\mathtt{dk} = z$ denotes the decryption key. Decryption is given by $g^m = \mathtt{Dec}^{\mathrm{EG}}_{\mathtt{dk}}(C) = e/u^z$. Let $\times$ denote element wise multiplication, e.g., $C_1 = (u_1, e_1), C_2 = (u_2, e_2)$, $C_1 \times C_2$ is defined as $(u_1 u_2, e_1 e_2)$.

- $\mathtt{PHash}_x^D$ is executed between S_1 and S_2. S_2 computes $h_{x,2} = (\mathtt{k}_{\mathtt{p}0}[1] \cdot \mathtt{k}_{\mathtt{p}0,2}^{\xi_2})^{r_2}$ and sends it to S_1. Eventually, S_1 holds $h_x = \mathtt{k}_{\mathtt{p}0,1}^{r_1 + r_2} \cdot \mathtt{k}_{\mathtt{p}0,2}^{\xi_1 \cdot r_1 + \xi_2 \cdot r_2}$. Note that S_1 always performs checks that $\mathtt{k}_{\mathtt{p}0} \in \mathbb{G}$ and $\mathbb{G} \ni h_2^x \neq 0$.
- $\mathtt{Hash}_0^D$ is executed between S_1 and S_2 such that S_1 eventually holds h_0. Let S_i for $i \in \{1, 2\}$ denote the participating party knowing $(\mathrm{pw}_i, \mathtt{sk}_i, \mathtt{k}_{\mathtt{h}i} = (\eta_{1,i}, \eta_{2,i}, \theta_i, \mu_i, \nu_i), \mathtt{pk}_1, \mathtt{pk}_2, C_0 = (u_{1,0}, u_{2,0}, e_0, v_0, \xi_0))$.
  - S_1 computes $m_0 \leftarrow \mathtt{Enc}^{\mathrm{EG}}_{\mathtt{pk}_1}(g_1^{-\mu_1}; r)$ and $c_1' \leftarrow \mathtt{Enc}^{\mathrm{EG}}_{\mathtt{pk}_1}(g_1^{\mathrm{pw}_1}; r')$, and sends (m_0, c_1') to S_2.
  - Receiving (m_0, c_1') from S_1, S_2 computes

$$m_1 \leftarrow (m_0)^{\mathrm{pw}_2} \times (c_1')^{-\mu_2} \times \mathtt{Enc}^{\mathrm{EG}}_{\mathtt{pk}_1}(g_1^{-\mu_2 \cdot \mathrm{pw}_2} \cdot u_{1,0}^{\eta_{1,2} + \xi_0 \eta_{2,2}} \cdot u_{2,0}^{\theta_2} \cdot e_0^{\mu_2} \cdot v_0^{\nu_2}; r'')$$

    and sends it to S_1.
  - Receiving m_1, S_1 computes the hash value

$$h_0 = g_1^{-\mu \cdot \mathrm{pw}_1} \cdot \mathtt{Dec}^{\mathrm{EG}}_{\mathtt{dk}_1}(m_1) \cdot u_{1,0}^{\eta_{1,1} + \xi_0 \eta_{2,1}} \cdot u_{2,0}^{\theta_1} \cdot e_0^{\mu_1} \cdot v_0^{\nu_1}.$$

D UC Functionalities

Functionality $\mathcal{F}_{\texttt{crs}}$

$\mathcal{F}_{\texttt{crs}}$ is parametrised by a distribution $\mathcal{D}$ and proceeds as follows:

NewValue: Upon input $(\texttt{NV}, \texttt{sid})$ choose a value $d \in_R \mathcal{D}$, send d back to the activating party and store the value if this is the first invocation. In any other case return the value d to the activating party.

Fig. 2: Ideal Functionality $\mathcal{F}_{\texttt{crs}}$

Functionality $\mathcal{F}_{\texttt{CA}}$

Registration: Upon receiving the first message $(\texttt{Register}, \texttt{sid}, v)$ from party $\mathcal{P}$, send $(\texttt{Registered}, \texttt{sid}, v)$ to the adversary; upon receiving $\texttt{ok}$ from the adversary, and if $\texttt{sid} = \mathcal{P}$ and this is the first request from $\mathcal{P}$, then record the pair $(\mathcal{P}, v)$.

Retrieve: Upon receiving a message $(\texttt{Retrieve}, \texttt{sid})$ from party $\mathcal{P}'$, send $(\texttt{Retrieve}, \texttt{sid}, \mathcal{P}')$ to the adversary, and wait for an $\texttt{ok}$ from the adversary. Then, if there is a recorded pair $(\texttt{sid}, v)$ output $(\texttt{Retrieve}, \texttt{sid}, v)$ to $\mathcal{P}'$. Otherwise output $(\texttt{Retrieve}, \texttt{sid}, \perp)$ to $\mathcal{P}'$.

Fig. 3: Ideal Functionality $\mathcal{F}_{\texttt{CA}}$

Functionality $\mathcal{F}_{\texttt{init}}$

$\mathcal{F}_{\texttt{init}}$, with fixed session identifier 0, runs in the universe with parties $\mathcal{U}$ and adversary $\mathcal{S}$. When called the first time, it sets $\texttt{Hist} = \emptyset$.

Init: Upon receiving $(\texttt{init}, 0, \langle P_i, \mathcal{P}, \mathcal{F} \rangle)$ from P_i, where $\mathcal{P} \subseteq \mathcal{U}$, execute the following:

1. Send $(\texttt{init}, 0, \langle P_i, \mathcal{P}, \mathcal{F} \rangle)$ to $\mathcal{S}$.
2. Upon receiving back $(\texttt{setId}, 0, \langle \texttt{sid}', P_i, \mathcal{P}, \mathcal{F} \rangle)$ from $\mathcal{S}$, do the following:
   (a) If $\texttt{sid}' \in \texttt{Hist}$, choose an arbitrary $\texttt{sid} \notin \texttt{Hist}$.
   (b) If $\texttt{sid}' \notin \texttt{Hist}$, set $\texttt{sid} \leftarrow \texttt{sid}'$.
   (c) Update $\texttt{Hist} \leftarrow \texttt{Hist} \cup \{\texttt{sid}\}$.
   (d) Send $(\texttt{invoke}, 0, \langle \texttt{sid}, P_i, \mathcal{P}, \mathcal{F} \rangle)$ to $\mathcal{S}$.
3. Upon receiving a message $(\texttt{sendoutput}, 0, \langle \texttt{sid}, P_i, \mathcal{P}, \mathcal{F} \rangle)$ from $\mathcal{S}$:
   (a) If $P_j \in \mathcal{P}$ and it has not yet been sent to the $\texttt{invoke}$ message with $\langle \texttt{sid}, P_i, \mathcal{P}, \mathcal{F} \rangle$), send it $(\texttt{invoke}, 0, \langle \texttt{sid}, P_i, \mathcal{P}, \mathcal{F} \rangle)$.

Fig. 4: Ideal Functionality $\mathcal{F}_{\texttt{init}}$

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

The functionality $\mathcal{F}_{\text{PAKE}}$ is parametrized by a security parameter λ. It interacts with an adversary $\texttt{SIM}$ and a set of parties via the following queries:

NewSession: Upon input $(\texttt{NS}, \texttt{sid}, P_i, P_j, \text{pw}, \texttt{role})$ from P_i, check that P_j is legit and send $(\texttt{NS}, \texttt{sid}, P_i, P_j, \texttt{role})$ to $\texttt{SIM}$. If this is the first NewSession query, or if this is the second NewSession query and there is a record $(\texttt{sid}, P_j, P_i, \text{pw}')$, then record $(\texttt{sid}, P_i, P_j, \text{pw})$ and mark this record fresh.

TestPwd: Upon input $(\texttt{TP}, \texttt{sid}, P_i, \text{pw}')$ from $\texttt{SIM}$, check that a fresh record $(\texttt{sid}, P_i, P_j, \text{pw})$ exists, then do: If $\text{pw} = \text{pw}'$, mark the record as $\texttt{compromised}$ and reply to $\texttt{SIM}$ with "correct guess". If $\text{pw} \neq \text{pw}'$, mark the record $\texttt{interrupted}$ and reply with "wrong guess".

NewKey: Upon input $(\texttt{NK}, \texttt{sid}, P_i, \texttt{sk})$ from $\texttt{SIM}$, check that a record $(\texttt{sid}, P_i, P_j, \text{pw})$ exists, $|\texttt{sk}| = \lambda$ and this is the first NewKey query for P_i, then:

- If the record is $\texttt{compromised}$, or either P_i or P_j is corrupted, then output $(\texttt{sid}, \texttt{sk})$ to P_i.
- If the record is fresh, and there is a record $(\texttt{sid}, P_j, P_i, \text{pw}')$ with $\text{pw}' = \text{pw}$, and a key $\texttt{sk}'$ was sent to P_j and $(\texttt{sid}, P_j, P_i, \text{pw})$ was fresh at the time, then output $(\texttt{sid}, \texttt{sk}')$ to P_i.
- In any other case, pick a new random key $\texttt{sk}'$ of length λ and send $(\texttt{sid}, \texttt{sk}')$ to P_i.

Either way, mark the record $(\texttt{sid}, P_i, P_j, \text{pw})$ as completed.

Fig. 5: Ideal Functionality $\mathcal{F}_{\text{PAKE}}$

E $\mathcal{F}_{\text{2PAKE}}$ Discussion

In this section we discuss some additional points of the $\mathcal{F}_{\text{2PAKE}}$ functionality and investigate relations to other 2PAKE security models and UC models in the password setting.

E.1 $\mathcal{F}_{\text{2PAKE}}$ and the BPR 2PAKE Model

While other security models for 2PAKE protocols where proposed [37], the BPR-like security model from [29] is the most comprehensible and (in its two-party version) established model. We therefore discuss relation between the proposed 2PAKE UC-security using $\mathcal{F}_{\text{2PAKE}}$ and the BPR-like security model from [29]. To compare security of a 2PAKE protocol Π in a game-based and UC setting we have to ensure that it supports session ids (necessary in the UC framework). We therefore assume that Π already uses UC compliant session ids. Note that it is easy to transform any 2PAKE protocol into a 2PAKE protocol with such session ids. Before looking into relation between the full game-based model for 2PAKE and $\mathcal{F}_{\text{2PAKE}}$ we want to point out that Π, securely realising $\mathcal{F}_{\text{2PAKE}}$, offers "forward secrecy", i.e. even an adversary that knows the correct password is not able to attack an execution of Π without actively taking part in the execution. With this in mind it is easy to see that Π, securely realising $\mathcal{F}_{\text{2PAKE}}$, is secure in the BPR-like model from [29]. This is because the attacker is either passive, which is covered by the previous observation, or is active and is therefore able tests one password. Those password tests ($\texttt{TestPwd}$ in $\mathcal{F}_{\text{2PAKE}}$ and $\texttt{Send}$ in the game based model) give the attacker a success probability of $q/|\mathcal{D}|$, with q the number of active sessions and $|\mathcal{D}|$ the dictionary size, when considering a uniform distribution of passwords inside the dictionary $\mathcal{D}$. Note that while the attacker may have knowledge of a password share, this does not increase this probability. Security on the model from [29] follows.

E.2 $\mathcal{F}_{\text{2PAKE}}$ and $\mathcal{F}_{\text{PAKE}}$

While $\mathcal{F}_{\text{PAKE}}$ and $\mathcal{F}_{\text{2PAKE}}$ are very similar they contain some significant difference we want to point out here. First, the key-exchange is performed between all three participants, but only C and, w.l.o.g., S_1 agree

on a common session key. The `role` is a technical necessity in $\mathcal{F}_{\text{PAKE}}$ for correct execution. Since we have explicit roles in $\mathcal{F}_{\text{2PAKE}}$ this is not necessary here. Due to the asymmetry in $\mathcal{F}_{\text{2PAKE}}$ (a client negotiates with two servers) we assume that the client is always the invoking party. While this is the case in $\mathcal{F}_{\text{PAKE}}$ as well when considering a real world scenario, the roles might be different there such that any of the two participating parties can start the protocol execution. The asymmetric setting in $\mathcal{F}_{\text{2PAKE}}$ further restricts `TestPwd` queries to the client since the servers hold high entropy password shares. While it is enough for the attacker to corrupt one party in $\mathcal{F}_{\text{PAKE}}$ to control the session key, in $\mathcal{F}_{\text{2PAKE}}$ he has to either corrupt or compromise the client, or corrupt both servers. As long as only one server is corrupted, the adversary has no control over the session keys and the parties receive uniformly at random chosen session keys In $\mathcal{F}_{\text{2PAKE}}$ session ids are human memorisable, consisting of all three involved parties (C, S_1, S_2), and unique query identifier is used to distinguish between different (possibly concurrent) protocol runs of one account (`sid`). This is a rather technical difference to $\mathcal{F}_{\text{PAKE}}$ that uses only session identifiers.

E.3 Corruptions

The two-server extension of the BPR 2PAKE model used in [29] does not consider corruptions at all. While parties can be malicious in the model (static corruption), the attacker is not allowed to query a corrupt oracle to retrieve passwords or internal state of participants. In our model the attacker is allowed to corrupt parties before execution. This however implies security in the model from [29] even if the attacker is allowed to corrupt clients to retrieve their passwords. This is because the environment can provide the BPR attacker with the password. However, this does not increase his success probability. Dynamic corruptions in $\mathcal{F}_{\text{2PAKE}}$ on the other hand are much more intricate. While UC-secure two party PAKE protocols with dynamic corruptions exist their approaches are not translatable to the 2PAKE setting. The challenge of dynamic corruptions is that the simulation has to be correct even if the attacker corrupts one party *after* the protocol execution has started. This is left open for future work.