Universally Composable Two-Server PAKE

Franziskus Kiefer[1(✉)] and Mark Manulis[2]

[1] Mozilla, Berlin, Germany
mail@franziskuskiefer.de
[2] Department of Computer Science, Surrey Center for Cyber Security, University of Surrey, Guildford, 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 [4,8] and the PAKE model in the Universal Composability (UC) framework by Canetti [18]. 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

M. Bishop and A.C.A. Nascimento (Eds.): ISC 2016, LNCS 9866, pp. 147–166, 2016.
DOI: 10.1007/978-3-319-45871-7_10

number of approaches have been proposed to alleviate this threat. For instance, verifier-based PAKE [11,22,34], 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 [23,31].

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. [12,27,29,33]) where both servers compute the same session key and asymmetric (e.g. [27]) where each server can compute an independent session key with the client or assist another server in the authentication process [26,35] 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. [30,32], that adopt t-out-of-n secret sharing, existing TPAKE protocols do not necessarily provide solutions for 2PAKE, e.g. [32] 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. [13,14,24], 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 [29] 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 [29], a generalisation of [27] 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) ([5,6] amongst others) holds in those groups. SXDH states that the DDH problem is hard in $\mathbb{G}_1$ and $\mathbb{G}_2$. 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. By $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com})$ we denote an efficient commitment scheme and use Pedersen commitments in our constructions where $(g, h, q, \lambda) \leftarrow \mathtt{CSetup}(\lambda)$ and $C \leftarrow \mathtt{Com} = (x; r) = g^x h^r$ with g and h being generators of a cyclic group $\mathbb{G}$ of prime-order q with bit-length in the security parameter λ and where the discrete logarithm of h with respect to base g is not known. Pedersen commitments are *additively homomorph*, i.e. for all $(C_i, d_i) \leftarrow \mathtt{Com}(x_i; r_i)$, $i \in 0, \ldots, m$ we have $\prod_{i=0}^{m} C_i = \mathtt{Com}(\sum_{i=0}^{m} x_i; \sum_{i=0}^{m} r_i)$.

Committed Zero-Knowledge Proofs. We use committed Σ-protocols for security against malicious verifiers [21,25]. Note that we do not require extractability (proof of knowledge) here, which allows us to avoid the necessity of rewinding. A zero-knowledge proof $\mathtt{ZKP}$ is executed between a prover and a verifier, proving that a word x is in a language L, using a witness w proving so.[1] 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 $C = (\ell, \boldsymbol{u}, e, v) \leftarrow \mathtt{Enc}_{\mathtt{pk}}^{\mathrm{CS}}(\ell, m; r)$ (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

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

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 $g_1^m = \mathtt{Dec}_{\mathtt{dk}}^{\mathrm{CS}}(C) = e/u_1^z$ 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 $C \leftarrow \mathtt{Enc}_{\mathtt{pk}}^{\mathcal{L}}(\ell, \mathrm{pw}; r)$.*

Smooth projective hashing for languages of ciphertexts where the projection key does not depend on the ciphertext is defined as follows (see also [10,28]).

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:*

- $\mathtt{KGen}_{\mathtt{H}}(L_{\mathrm{pw}})$ *generates a random hashing key* $\mathtt{k_h}$ *for language* L_{pw}.
- $\mathtt{KGen}_{\mathtt{P}}(\mathtt{k_h}, L_{\mathrm{pw}})$ *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.

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'.[2]

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.*

2.3 Distributed Smooth Projective Hashing

Another flavor, called Distributed SPHF (D-SPHF), was introduced in [29] for use in (non-composable) 2PAKE protocols such as [27] where servers hold password shares pw_1 and pw_2 respectively, and the client holds $\mathrm{pw} = \mathrm{pw}_1 + \mathrm{pw}_2$. 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 $i \in \{0, 1, 2\}$.*
- $\mathtt{Hash}_x(\mathtt{k_{h0}}, L_{\widehat{\mathrm{pw}}}, C_1, C_2)$ *computes hash value h_x from hashing key* $\mathtt{k_{h0}}$ *and two server ciphertexts C_1 and C_2.*
- $\mathtt{PHash}_x(\mathtt{k_{p0}}, L_{\widehat{\mathrm{pw}}}, C_1, C_2, r_1, r_2)$ *computes hash value h_x from projection key* $\mathtt{k_{p0}}$*, two ciphertexts C_1 and C_2, and witnesses r_1 and r_2.*

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

– $\mathtt{Hash}_0(\mathtt{k}_{\mathtt{h}1}, \mathtt{k}_{\mathtt{h}2}, L_{\widehat{\mathrm{pw}}}, C_0)$ *computes hash value* h_0 *from hashing keys* $\mathtt{k}_{\mathtt{h}1}$ *and* $\mathtt{k}_{\mathtt{h}2}$ *and ciphertext* C_0.
– $\mathtt{PHash}_0(\mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2}, L_{\widehat{\mathrm{pw}}}, C_0, r_0)$ *computes hash value* h_0 *from projection keys* $\mathtt{k}_{\mathtt{p}1}$ *and* $\mathtt{k}_{\mathtt{p}2}$*, 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 $\mathtt{Setup}$, $\mathtt{PHash}_x^D$ and $\mathtt{Hash}_0^D$. Let Π denote D-SPHF as described above.

– $\mathtt{Setup}(\mathrm{pw}, \mathrm{pw}_1, \mathrm{pw}_2, C, S_1, S_2)$ initialises a new instance for each participant with $(\mathrm{pw}, C, S_1, S_2)$ for C, $(\mathrm{pw}_1, S_1, C, S_2)$ for S_1 and $(\mathrm{pw}_2, S_2, C, S_1)$ for S_2. Eventually, all participants compute and broadcast projection keys $\mathtt{k}_{\mathtt{p}i}$ and encryptions $C_i \leftarrow \mathtt{Enc}_{\mathtt{pk}}^{\mathcal{L}}(\ell_i, \mathrm{pw}_i; r_i)$ of their password (share) pw_i using $\Pi.\mathtt{KGen}_{\mathtt{H}}$, $\Pi.\mathtt{KGen}_{\mathtt{P}}$ and the associated encryption scheme $\mathcal{L}$. Participants store incoming $\mathtt{k}_{\mathtt{p}i}, C_i$ for later use. After receiving $(\mathtt{k}_{\mathtt{p}1}, C_1, \mathtt{k}_{\mathtt{p}2}, C_2)$, the client computes $h_0 \leftarrow \Pi.\mathtt{PHash}_0(\mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2}, L_{\widehat{\mathrm{pw}}}, C_0, r_0)$ and $h_x \leftarrow \Pi.\mathtt{Hash}_x(\mathtt{k}_{\mathtt{h}0}, L_{\widehat{\mathrm{pw}}}, C_1, C_2)$.
– $\mathtt{PHash}_x^D$ is executed between S_1 and S_2. Each server S_i performs $\mathtt{PHash}_x^D$ on input $(\mathtt{k}_{\mathtt{p}0}, \mathrm{pw}_i, C_1, C_2, r_i)$ such that S_1 eventually holds h_x while S_2 learns nothing about h_x.
– $\mathtt{Hash}_0^D$ is executed between S_1 and S_2. Each server S_i performs $\mathtt{Hash}_0^D$ on input $(\mathrm{pw}_i, \mathtt{k}_{\mathtt{h}i}, C_0, C_1, C_2)$ such that S_1 eventually holds h_0 while S_2 learns nothing about h_0.

2.4 Ideal Functionalities

For our 2PAKE realisation we rely on some commonly used ideal functionalities within the UC framework. These are: $\mathcal{F}_{\mathtt{crs}}$ for the common reference string from [17], $\mathcal{F}_{\mathtt{CA}}$ for the CA from [16] to establish verified public keys for the servers, $\mathcal{F}_{\mathtt{init}}$ from [7] to establish unique query identifiers between the parties in a protocol. We refer for their descriptions to the original sources.

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. Sect. 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 [29].

Definition 5 (TD-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 trapdoor distributed smooth projective hash function for language* $L_{\widehat{\mathrm{pw}}}$ *consists of the following ten algorithms:*

- $(\mathtt{crs}', \tau') \xleftarrow{R} \mathtt{TSetup}(\mathtt{crs})$ *generates* $\mathtt{crs}'$ *with trapdoor* τ' *from* $\mathtt{crs}$
- $\mathtt{KGen_H}$, $\mathtt{KGen_P}$, $\mathtt{Hash}_x$, $\mathtt{PHash}_x$, $\mathtt{Hash}_0$, $\mathtt{PHash}_0$ *behave as for D-SPHF*
- $b \leftarrow \mathtt{VerKp}(\mathtt{k_p}, L_{\widehat{\mathrm{pw}}})$ *returns* $b = 1$ *iff* $\mathtt{k_p}$ *is a valid projection key and* $b = 0$ *otherwise*
- $h_x \leftarrow \mathtt{THash}_x(\mathtt{k_{p_0}}, L_{\widehat{\mathrm{pw}}}, C_1, C_2, \tau')$ *computes hash value* h_x *of ciphertexts* C_1 *and* C_2 *using projection key* $\mathtt{k_{p_0}}$ *and trapdoor* τ'
- $h_0 \leftarrow \mathtt{THash}_0(\mathtt{k_{p_1}}, \mathtt{k_{p_2}}, L_{\widehat{\mathrm{pw}}}, C_0, \tau')$ *computes hash value* h_0 *of* C_0 *using projection keys* $\mathtt{k_{p_1}}$ *and* $\mathtt{k_{p_2}}$*, 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 $(\mathtt{k_{h0}}, \mathtt{k_{h1}}, \mathtt{k_{h2}})$ and $(\mathtt{k_{p0}}, \mathtt{k_{p1}}, \mathtt{k_{p2}})$, it holds not only that

$$\mathtt{Hash}_0(\mathtt{k_{h1}}, \mathtt{k_{h2}}, L_{\widehat{\mathrm{pw}}}, C_0) = \mathtt{PHash}_0(\mathtt{k_{p_1}}, \mathtt{k_{p_2}}, L_{\mathrm{pw},\mathrm{pw}_1,\mathrm{pw}_2}, C_0, r_0), \text{ and}$$
$$\mathtt{Hash}_x(\mathtt{k_{h0}}, L_{\widehat{\mathrm{pw}}}, C_1, C_2) = \mathtt{PHash}_x(\mathtt{k_{p_0}}, L_{\mathrm{pw},\mathrm{pw}_1,\mathrm{pw}_2}, C_1, C_2, r_1, r_2)$$

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

$$\mathtt{Hash}_0(\mathtt{k_{h1}}, \mathtt{k_{h2}}, L_{\widehat{\mathrm{pw}}}, C_0) = \mathtt{THash}_0(\mathtt{k_{p_1}}, \mathtt{k_{p_2}}, L_{\mathrm{pw},\mathrm{pw}_1,\mathrm{pw}_2}, C_0, \tau') \text{ and}$$
$$\mathtt{Hash}_x(\mathtt{k_{h0}}, L_{\widehat{\mathrm{pw}}}, C_1, C_2) = \mathtt{THash}_x(\mathtt{k_{p_0}}, L_{\mathrm{pw},\mathrm{pw}_1,\mathrm{pw}_2}, C_1, C_2, \tau').$$

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* $\mathtt{k_p}$*, 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* $(\mathtt{k_{p_0}}, \mathtt{k_{p_1}}, \mathtt{k_{p_2}})$ *are valid, i.e.* $\mathtt{VerKp}(\mathtt{k_{p}}_i, L_{\widehat{\mathrm{pw}}}) = 1$ *for* $i \in \{0, 1, 2\}$*, but*

$$\mathtt{THash}_x(\mathtt{k_{p_0}}, L_{\widehat{\mathrm{pw}}}, C_1, C_2, \tau') \neq \mathtt{PHash}_x(\mathtt{k_{p_0}}, L_{\widehat{\mathrm{pw}}}, C_1, C_2, r_1, r_2) \text{ or}$$
$$\mathtt{THash}_0(\mathtt{k_{p_1}}, \mathtt{k_{p_2}}, L_{\widehat{\mathrm{pw}}}, C_0, \tau') \neq \mathtt{PHash}_0(\mathtt{k_{p_1}}, \mathtt{k_{p_2}}, L_{\widehat{\mathrm{pw}}}, C_0, r_0)$$

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 [29] 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 Sect. 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 $h_x = e(h'_x, g_2)$
- $\mathtt{Hash}_0(\mathtt{k}_{\mathtt{h}1}, \mathtt{k}_{\mathtt{h}2}, L_{\widehat{\mathrm{pw}}}, C_0)$ computes

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

  and outputs $h_0 = e(h'_0, g_2)$
- $\mathtt{PHash}_0(\mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2}, L_{\widehat{\mathrm{pw}}}, C_0, r_0)$ computes

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

  and outputs $h_0 = e(h'_0, g_2)$
- $\mathtt{VerKp}(\mathtt{k}_{\mathtt{p}i}, L_{\widehat{\mathrm{pw}}})$ verifies that

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

  and

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

- $\mathtt{THash}_0(\mathtt{k}_{\mathtt{p}1}, \mathtt{k}_{\mathtt{p}2}, L_{\widehat{\mathrm{pw}}}, C_0, \tau')$ computes

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

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

- $\mathtt{THash}_x(\mathtt{k}_{\mathtt{p}0}, L_{\widehat{\mathrm{pw}}}, C_1, C_2, \tau')$ computes

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

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

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. Sect. 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 $(\texttt{sid}, \texttt{qid}, 0, Q, C_1, \texttt{k}_{\texttt{p}_1})$ and $(\texttt{sid}, \texttt{qid}, 0, R, C_2, \texttt{k}_{\texttt{p}_2})$ it proceeds as follows. P proceeds only if the projection keys $\texttt{k}_{\texttt{p}_1}$ and $\texttt{k}_{\texttt{p}_2}$ are correct, i.e. $\texttt{VerKp}(\texttt{k}_{\texttt{p}_1}, L_{\widehat{\text{pw}}}) = 1$ and $\texttt{VerKp}(\texttt{k}_{\texttt{p}_2}, L_{\widehat{\text{pw}}}) = 1$. If the verification fails, P outputs $(\texttt{sid}, \texttt{qid}, \perp, \perp)$ and aborts the protocol.

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

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

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

(c) On input $(\texttt{sid}, \texttt{qid}, \texttt{PHash}_x, 0, S_2, C_{h_{x,2}})$ S_1 in the correct state draws challenge $\texttt{c} \in_R \mathbb{Z}_q$ and returns $(\texttt{sid}, \texttt{qid}, \texttt{PHash}_x, 1, S_1, \texttt{c})$ 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 $h_x = e\left(h_{x,2} \cdot (\mathtt{k}_{\mathtt{p}0,1} \cdot \mathtt{k}_{\mathtt{p}0,2}^{\xi_1})^{r_1}, g_2\right)$ if the verifications are successful, $h_x \neq \perp$ and $h_0 \neq \perp$, or sets $h_0 = \perp$ and $h_x = \perp$ 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 $C_{\mathtt{Rs}2} = g_{1,1}^{H(\mathtt{Rs}_2)} h^{r_{c4}}$ with $r_{c4} \in_R \mathbb{Z}_q$ and sends $(\mathtt{sid}, \mathtt{qid}, \mathtt{Hash}_{0,3}, S_2, C_{\mathtt{Rs}2})$ 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 $(t_{\overline{m}1}, t_{\overline{m}2}, t_{e2}, t_{v2}, t_{\mathtt{k}_\mathtt{p}12}, t_{\mathtt{k}_\mathtt{p}22})$ 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 `crs`. The construction of the according Σ proof is straight-forward. The prover computes commitments

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

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 [29] 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 [29] 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 Sect. 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 Fig. 1. Observe that implicit client authentication is sufficient for building UC-secure channels [19]. 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 [18], 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. [1–3], rely on more complex SPHF constructions that are not translatable to the distributed setting of D-SPHF.

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

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

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

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

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

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

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

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

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

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 $\mathcal{F}_{\mathtt{init}}$. 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 Sect. 4.1 securely realises $\widehat{\mathcal{F}}_{\text{2PAKE}}$ with static corruptions in the $\mathcal{F}_{\mathtt{crs}}$-$\mathcal{F}_{\mathtt{CA}}$-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.*

Proof (Sketch). In the following we highlight changes in the sequence of games from the real-world execution in $\mathcal{G}_1$ to the ideal-world execution via $\mathcal{F}_{\text{2PAKE}}$ in $\mathcal{G}_{17}$ and describe the ideal-world adversary `SIM`. Due to space limitations the analysis of game hops is available in the full version.

$\mathcal{G}_1$: 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.

$\mathcal{G}_2$: In this game all honest participants are replaced by a challenger $\mathcal{C}$ that generates `crs` together with its trapdoor τ and interacts with $\mathcal{A}$ on behalf of honest parties.

$\mathcal{G}_3$: 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. $\text{pw}' = \text{pw}$. If this is not the case, $\text{pw}' \neq \text{pw}$, $\mathcal{C}$ 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.

$\mathcal{G}_4$: In this game $\mathcal{C}$ chooses $\mathtt{sk} \in_R \mathbb{G}_T$ at random if h_0 was chosen at random (as in $\mathcal{G}_0$) and computation of `sk` on S_1 is successful.

$\mathcal{G}_5$: Upon receiving an adversarially generated 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.

$\mathcal{G}_6$ **:** In this game $\mathcal{C}$ chooses $\mathtt{sk} \in_R \mathbb{G}_T$ at random if h_x was chosen at random (as in $\mathcal{G}_0$) and computation of $\mathtt{sk}$ on C is successful, i.e., projection keys $\mathtt{k}_{\mathtt{p}_1}$ and $\mathtt{k}_{\mathtt{p}_2}$ are correct.

$\mathcal{G}_7$ **:** $\mathcal{C}$ replaces 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.

$\mathcal{G}_8$ **:** 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 $h_0 \in_R \mathbb{G}_T$.

$\mathcal{G}_9$ **:** In this game $\mathcal{C}$ chooses $\mathtt{sk} \in_R \mathbb{G}_T$ at random in case h_0 was chosen at random (as in $\mathcal{G}_0$) and computation of $\mathtt{sk}$ on S_1 is successful.

$\mathcal{G}_{10}$ **:** Upon 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 $\mathtt{Hash}_x$.

$\mathcal{G}_{11}$ **:** In this game $\mathcal{C}$ chooses $\mathtt{sk} \in_R \mathbb{G}_T$ at random in case h_0 was chosen random (as in $\mathcal{G}_0$) and computation of $\mathtt{sk}$ on C is successful (projection keys $\mathtt{k}_{\mathtt{p}_1}$ and $\mathtt{k}_{\mathtt{p}_2}$ are correct).

$\mathcal{G}_{12}$ **:** The entire $\mathtt{crs}$ including ζ is chosen now by challenger $\mathcal{C}$.

$\mathcal{G}_{13}$ **:** 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'.

$\mathcal{G}_{14}$ **:** 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 possible because $\mathcal{C}$ now knows trapdoor τ'.

$\mathcal{G}_{15}$ **:** 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).

$\mathcal{G}_{16}$ **:** 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 $\mathrm{pw}'_i \in_R \mathbb{Z}_q$.

$\mathcal{G}_{17}$ **:** This is the final game where instead of the challenger $\mathcal{C}$ the simulation is done by the ideal-world adversary (simulator) $\mathtt{SIM}$ that further interacts with the ideal functionality $\mathcal{F}_{\mathrm{2PAKE}}$. While this game is structurally different from $\mathcal{G}_0$ the interaction with $\mathcal{A}$ is indistinguishable from the latter. This combined with the following description of the simulator concludes the proof.

Simulator. We describe $\mathtt{SIM}$ for a single session $\mathtt{sid} = (C, S_1, S_2)$. The security then follows from the composition theorem [15] covering multiple sessions and from the joint-state composition theorem [20], covering creation of a joint state by $\mathcal{F}_{\mathtt{CA}}$ and $\mathcal{F}_{\mathtt{crs}}$ for all sessions and participants. As before, we assume that 0 is not a valid password.

First, $\mathtt{SIM}$ generates $\mathtt{crs} = (q, g_{1,1}, g_{1,2}, h, c, d, \mathbb{G}_1, g_2, \zeta, \mathbb{G}_2, \mathbb{G}_T, e, H_k)$ 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.

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 party P by computing $M_i = (C_i, \texttt{k}_{\texttt{p}_i})$ for $i \in \{0, 1, 2\}$ and 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 $(C_i, \texttt{k}_{\texttt{p}_i})$ to $\mathcal{A}$. The first round of messages is handled as follows.

(i) When a 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 a 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 a 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 $\mathcal{F}_{\text{2PAKE}}$ 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, $\texttt{SIM}$ proceeds as follows. After C received all ciphertexts and projection keys 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 $\texttt{sk}_C$ for this session exists, or $(\texttt{NK}, \texttt{sid}, \texttt{qid}, C, \perp)$ otherwise. After S_1 and S_2 received all ciphertexts and projection keys and the previously described checks were performed, $\texttt{SIM}$ simulates $\texttt{PHash}_x$ and $\texttt{Hash}_0$ computations between S_1 and S_2 with random elements and simulated zero-knowledge proofs. If all messages received by S_1 are oracle generated, $\texttt{SIM}$ sends $(\texttt{NK}, \texttt{sid}, \texttt{qid}, S_1, \texttt{sk}_S)$ 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, SIM does nothing and aborts the session for S_1.

5 $\mathcal{F}_{\text{2PAKE}}$ Discussion

$\mathcal{F}_{\text{2PAKE}}$ and the BPR 2PAKE Model. While other security models for 2PAKE protocols where proposed [33], the BPR-like security model from [27] is the most comprehensible and (in its two-party version) established model. 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. Before looking into the relation between the 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 [27]. 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 (TestPwd in $\mathcal{F}_{\text{2PAKE}}$ and 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 $\mathcal{D}$.

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

Corruptions. The two-server extension of the BPR 2PAKE model used in [27] 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 [27] 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.

6 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. Abdalla, M., Benhamouda, F., Blazy, O., Chevalier, C., Pointcheval, D.: SPHF-friendly non-interactive commitments. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 214–234. Springer, Heidelberg (2013)
2. Abdalla, M., Benhamouda, F., Pointcheval, D.: Removing Erasures with Explainable Hash Proof Systems. Cryptology ePrint Archive, Report 2014/125 (2014)
3. Abdalla, M., Chevalier, C., Pointcheval, D.: Smooth projective hashing for conditionally extractable commitments. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 671–689. Springer, Heidelberg (2009)
4. Abdalla, M., Fouque, P.-A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (2005)
5. Ateniese, G., Camenisch, J., Hohenberger, S., de Medeiros, B.: Practical group signatures without random oracles. Cryptology ePrint Archive, 2005:385 (2005)
6. Ballard, L., Green, M., de Medeiros, B., Monrose, F.: Correlation-resistant storage via keyword-searchable encryption. Cryptology ePrint Archive, 2005:417 (2005)
7. Barak, B., Lindell, Y., Rabin, T.: Protocol Initialization for the Framework of Universal Composability. Cryptology ePrint Archive, 2004:6 (2004)
8. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
9. Bellovin, S.M., Merritt, M.: Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise. In: ACM CCS 1993, pp. 244–250. ACM (1993)
10. Benhamouda, F., Blazy, O., Chevalier, C., Pointcheval, D., Vergnaud, D.: New techniques for SPHFs and efficient one-round PAKE protocols. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 449–475. Springer, Heidelberg (2013)

11. Benhamouda, F., Pointcheval, D.: Verifier-based password-authenticated key exchange: New models and constructions. Cryptology ePrint Archive, 2013:833 (2013)
12. Brainard, J., Juels, A.: A new two-server approach for authentication with short secrets. In: USENIX03 (2003)
13. Camenisch, J., Enderlein, R.R., Neven, G.: Two-Server Password-Authenticated Secret Sharing UC-Secure Against Transient Corruptions. Cryptology ePrint Archive, 2015:006 (2015)
14. Camenisch, J., Lysyanskaya, A., Neven, G.: Practical yet universally composable two-server password-authenticated secret sharing, pp. 525–536. ACM (2012)
15. Canetti, R., Security, U.C.: A new paradigm for cryptographic protocols. In: FOCS 2001, p. 136. IEEE CS, Washington, DC, USA (2001)
16. Canetti, R.: Universally composable signature, certification, and authentication. In: CSFW 2004, p. 219. IEEE CS (2004)
17. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001)
18. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005)
19. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
20. Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (2003)
21. Damgård, I.B.: Efficient concurrent zero-knowledge in the auxiliary string model. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 418–430. Springer, Heidelberg (2000)
22. Gentry, C., MacKenzie, P.D., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 142–159. Springer, Heidelberg (2006)
23. hashcat. hashcat - advanced password recovery (2014). http://hashcat.net/. Accessed 1 Dec 2014
24. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 233–253. Springer, Heidelberg (2014)
25. Jarecki, S., Lysyanskaya, A.: Adaptively secure threshold cryptography: introducing concurrency, removing erasures (Extended Abstract). In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 221–242. Springer, Heidelberg (2000)
26. Jin, H., Wong, D.S., Xu, Y.: An efficient password-only two-server authenticated key exchange system. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 44–56. Springer, Heidelberg (2007)
27. Katz, J., MacKenzie, P.D., Taban, G., Gligor, V.D.: Two-server password-only authenticated key exchange. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 1–16. Springer, Heidelberg (2005)
28. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 293–310. Springer, Heidelberg (2011)

29. Kiefer, F., Manulis, M.: Distributed smooth projective hashing and its application to two-server password authenticated key exchange. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 199–216. Springer,
Heidelberg (2014)
30. MacKenzie, P., Shrimpton, T., Jakobsson, M.: Threshold password-authenticated key exchange. In: CRYPTO 2002, p. 141 (2002)
31. Openwall. John the Ripper password cracker (2014). http://www.openwall.com/john/. Accessed 1 Dec 2014
32. Raimondo, M.D., Gennaro, R.: Provably secure threshold password-authenticated key exchange. In: EUROCRYPT 2003, p. 507523 (2003)
33. Szydlo, M., Kaliski, B.: Proofs for two-server password authentication. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 227–244. Springer, Heidelberg (2005)
34. Wu, T.: RFC 2945 - The SRP Authentication and Key Exchange System, September 2000
35. Yang, Y., Deng, R., Bao, F.: A practical password-based two-server authentication and key exchange system. IEEE TDSC **3**(2), 105–114 (2006)