Mitigating Server Breaches in Password-Based Authentication: Secure and Efficient Solutions

Olivier Blazy[1], Céline Chevalier[2], and Damien Vergnaud[3]

[1] Université de Limoges, XLim, France
[2] Université Panthéon-Assas, Paris, France
[3] ENS, CNRS, INRIA, and PSL Research University, Paris, France

Abstract. *Password-Authenticated Key Exchange* allows users to generate a strong cryptographic key based on a shared "human-memorable" password without requiring a public-key infrastructure. It is one of the most widely used and fundamental cryptographic primitives. Unfortunately, mass password theft from organizations is continually in the news and, even if passwords are salted and hashed, brute force breaking of password hashing is usually very successful in practice.

In this paper, we propose two efficient protocols where the password database is somehow shared among two servers (or more), and authentication requires a distributed computation involving the client and the servers. In this scenario, even if a server compromise is doable, the secret exposure is not valuable to the adversary since it reveals only a share of the password database and does not permit to brute force guess a password without further interactions with the parties for each guess. Our protocols rely on *smooth projective hash functions* and are proven secure under classical assumption in the standard model (*i.e.* do not require idealized assumption, such as random oracles).

Keywords. Password-Authenticated Key Exchange, Distributed Computation, Decision Diffie-Hellman, Smooth Projective Hashing

1 Introduction

Authenticated Key Exchange protocols enable two parties to establish a shared cryptographically strong key over an insecure network under the complete control of an adversary. This primitive is one of the most widely used and fundamental cryptographic primitives and it obviously requires the parties to have authentication means, *e.g.* (public or secret) cryptographic keys or short (*i.e.*, low-entropy) secret keys.

PAKE, for *Password-Authenticated Key Exchange*, allows users to generate a strong cryptographic key based on a shared "human-memorable" password without requiring a public-key infrastructure. In this setting, an adversary controlling all communication in the network should not be able to mount an *offline dictionary attack*. More precisely, an eavesdropper should not obtain enough information to be able to brute force guess a password without further interactions with the parties for each guess. Note that *online dictionary attacks* in which an adversary simply attempts to log-in repeatedly, trying each possible low-entropy password can be dealt with using other computer security methods (such as limiting the number of attempts). In particular, strong security can be obtained even using passwords chosen from a small set of possible values (a four-digit pin, for example).

Incidents of sensitive customer information "hacking" (including leaking of passwords) in e-commerce systems are frequently revealed in the newspaper. In addition to major reputational damage, a company with a significant data breach may be sued by its clients for the breach and may be suspended or disqualified from future public sector or government work.

To alleviate the threat that stored passwords are revealed immediately in case of a server compromise, many servers adopt the approach for storing passwords in a hashed form with a random salt. When the database of hashed password is compromised, the offline dictionary attack requires a more important computational effort but remains usually possible. The notion of *Verifier-based PAKE*, where the client owns a password pw and the server knows a one-way transformation v of the password only were proposed by Bellovin and Merritt [BM92]. The two players eventually agree on a common high entropy secret if and only if pw and v match together. It prevents massive password recovering in case of server corruption and it forces the attacker who breaks into the server and is willing to recover passwords to perform an additional costly offline dictionary attack.

We consider an alternative approach inspired by the multi-party computation paradigm (and first suggested by Ford and Kaliski [FK00]). The password database on the server side is somehow shared among two servers (or more, but we focus here on two for sake of simplicity), and authentication requires a distributed computation involving the client – who still does not need an additional cryptographic device capable of storing high-entropy secret keys – and the two servers who will use some additional shared secret information. The interaction is performed using a *gateway* that does not know any secret information and ends up in the gateway and the client sharing a common key. The lifetime of the protocol is divided into distinct periods (for simplicity, one may think of these time periods as being of equal length; e.g. one day) and at the beginning of each period, the two servers interact and update their sharing of the password database. Similarly to proactive schemes in multi-party computation, we allow the adversary multiple corruptions of each server, limiting only the corruptions to one server for each period. The user does not need to update his password nor to perform any kind of computations and its interaction with the two servers (performed using the gateway) remains the same for the lifetime of the protocol. In this scenario, even if a server compromise is doable, the secret exposure is not valuable to the adversary since it reveals only a share of the password database and does not permit to run an offline dictionary attack.

The goal of our paper is to present practical realizations based on classical cryptographic assumptions in the standard security model.

Related work. EKE (for Encrypted Key Exchange) is the most famous instantiation of *Password-Authenticated Key Exchange*. It has been proposed by Bellovin and Merritt [BM92] and simply consists of a Diffie-Hellman key exchange [DH76], where the flows are symmetrically encrypted under the shared password.

A first formal security model was proposed by Bellare, Pointcheval and Rogaway [BPR00] (the BPR model), to deal with offline dictionary attacks. It essentially says that the best attack should be the online exhaustive search, consisting in trying all the passwords by successive executions of the protocol with the server. Several variants of EKE with BPR-security proofs have been proposed in the ideal-cipher model or the random-oracle model (see the survey [Poi12] for details). Katz, Ostrovsky and Yung [KOY01] proposed the first practical scheme, provably secure in the standard model under the Decision Diffie-Hellman assumption (DDH). It has been generalized by Gennaro and Lindell [GL03], making use of smooth projective hash functions.

As mentioned above, Ford and Kaliski [FK00] were the first to propose to distribute the capability to test passwords over multiple servers. Building on this approach, several such protocols were subsequently proposed in various settings (*e.g.* [Jab01,MSJ02,BJKS03,DG03,DG06,SK05,KMTG05,KMTG12, ACFP05, KM14]) and it is worth noting that the protocol from [BJKS03] is commercially available as EMC's *RSA Distributed Credential Protection.* Recently, Camenisch, Enderlein and Neven [CEN15] revisited this approach and proposed a scheme in the universal composability framework [Can01] (which has obvious advantages for password-based protocols since users often use related passwords for many providers). Camenisch *et al.* gave interesting details about the steps that need to be taken when a compromise actually occurs. Unfortunately, due to the inherent difficulties of construction of the simulator in the universal composability framework, their scheme is inefficient since users and servers have to perform a few hundred exponentiations each.

Our contributions. In order to achieve practical constructions in the standard security model, we consider the variant of the BPR model[1] in the distributed setting proposed by Katz, MacKenzie, Taban and Gligor in [KMTG12]. In this security model, we assume that the communication between the client

[1] Our schemes can be adapted to achieve security in universal composability framework using techniques similar to those used in [CEN15]. The resulting schemes are slightly more efficient but are unfortunately still not practical.

and the authentication servers, is carried on a basically insecure network. Messages can be tapped and modified by an adversary and the communication between the clients and the servers is asynchronous. The adversary should not be able to brute force guess a password without further interactions with the client for each guess even if he corrupts and impersonates a server in an active way.

Our first construction uses a similar approach to the schemes from [Jab01, MSJ02, BJKS03, DG06, SK05, KMTG12, ACFP05, KM14]: the user generates information theoretic shares of his password and sends them to the servers. In the authentication phase, the parties run a dedicated protocol to verify that the provided password equals the priorly shared one. Our solution then consists in some sort of three-party PAKE, in which (1) the user implicitly checks (using a smooth projective hash function) that its password is indeed the sum of the shares owned by the two servers, and (2) each server implicitly checks that its share is the difference of the password owned by the user and the share owned by the other server. Contrary to the popular approach initiated in [KOY01, GL03] for PAKE, we cannot use two smooth projective hash functions (one for the client and one for the server) so we propose a technique in order to combine in a secure way six smooth projective hash functions. This new method (which may be of independent interest) allows us to prove the security of this construction under classical cryptographic assumptions (namely the DDH assumption) in the standard security model from [KMTG12] (without any idealized assumptions).

The main weakness of this first solution is that at each time period, the servers have to refresh the information-theoretic sharing of the password of all users. This can be handled easily using well-known techniques from proactive multi-party computation but if the number of users is large, this can be really time-consuming (in particular if the time period is very short). Our second construction (which is the main contribution of the paper) is built on the ideas from the first one but passwords are now encrypted using a public-key encryption scheme where the corresponding secret key is shared among the servers. At the beginning of each time period, the servers only need to refresh the sharing of this secret key but the password database is not modified (and can actually be public). Password verification and the authenticated key exchange is then carried out without ever decrypting the database. A secure protocol is run to verify that the password sent by the user matches the encrypted password. It is similar to the protocol we design for the first construction except that the user encrypts its password and the parties implicitly check (using in this case five smooth projective hash functions) that the message encrypted in this ciphertext is the same as the message encrypted in the database (using the secret key shared upon the servers). Both constructions consist in only two flows (one from the client and one from the servers) and a (private) flow from the servers to the gateway.

2 Preliminaries

In this section we recall various classical definitions, tools used throughout this paper. We use classical notions and notations and the familiar reader may skip this section.

Public-Key Encryption Scheme. An encryption scheme $\mathcal{E}$ is described through four algorithms (Setup, KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$):

- $\mathsf{Setup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}(\mathsf{param})$ outputs a pair of keys, a (public) encryption key ek and a (private) decryption key dk;
- $\mathsf{Encrypt}(\mathsf{ek}, M; \rho)$ outputs a ciphertext $\mathcal{C}$, on the message M, under the encryption key ek, with randomness ρ;
- $\mathsf{Decrypt}(\mathsf{dk}, \mathcal{C})$ outputs the plaintext M, encrypted in the ciphertext $\mathcal{C}$ or $\perp$.

Such encryption scheme is required to have the classical properties, *Correctness* and *Indistinguishability under Chosen Plaintext/Ciphertext Attack* [GM84]:

- *Correctness*: For every pair of keys $(\mathsf{ek}, \mathsf{dk})$ generated by KeyGen, every messages M, and every random ρ, we should have $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{Encrypt}(\mathsf{ek}, M; \rho)) = M$.

- *Indistinguishability under Chosen Plaintext Attack [GM84]*: This notion ($\mathsf{IND\text{-}CPA}$), formalized by the adjacent game, states that an adversary shouldn't be able to efficiently guess which message has been encrypted even if he chooses the two original plaintexts.
  The advantages are:

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}) = |\Pr[\mathsf{Exp}^{\mathsf{ind}-1}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}-0}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}) = 1]|$$

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(\mathfrak{K}, t) = \max_{\mathcal{A} \leq t} \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}).$$

```
$\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$
$\mathsf{param} \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
$(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
$(M_0, M_1) \leftarrow \mathcal{A}(\mathtt{FIND} : \mathsf{pk})$
$c^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}, M_b)$
$b' \leftarrow \mathcal{A}(\mathtt{GUESS} : c^*)$
RETURN $b'$
```

One might want to increase the requirements on the security of an encryption, in this case the $\mathsf{IND\text{-}CPA}$ notion can be strengthened into Indistinguishability under Adaptive Chosen Ciphertext Attack $\mathsf{IND\text{-}CCA}$. The non-adaptive notion was introduced in [NY90], while the adaptive one was in [RS92]:

- $\mathsf{IND\text{-}CCA}$: This notion states that an adversary should not be able to efficiently guess which message has been encrypted even if he chooses the two original plaintexts, and can ask several decryption of ciphertexts as long as they are not the challenge one.

```
$\mathsf{Exp}^{\mathsf{ind\text{-}cca}-b}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$
$\mathsf{param} \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
$(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
$(M_0, M_1) \leftarrow \mathcal{A}(\mathtt{FIND} : \mathsf{pk}, \mathsf{ODecrypt}(\cdot))$
$c^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}, M_b)$
$b' \leftarrow \mathcal{A}(\mathtt{GUESS} : c^*, \mathsf{ODecrypt}(\cdot))$
IF $(c^*) \in \mathcal{CT}$ RETURN 0
ELSE RETURN $b'$
```

  - Where the $\mathsf{ODecrypt}$ oracle outputs the decryption of c under the challenge decryption key dk. The input queries (c) are added to the list $\mathcal{CT}$ of decrypted ciphertexts.

Smooth Projective Hash Functions. Smooth Projective Hash Functions [CS02] were introduced by Cramer and Shoup. A projective hashing family is a family of hash functions that can be evaluated in two ways: using the (secret) hashing key, one can compute the function on every point in its domain, whereas using the (public) *projected* key one can only compute the function on a special subset of its domain. Such a family is deemed *smooth* if the value of the hash function on any point outside the special subset is independent of the projected key. The notion of SPHF has found applications in various contexts in cryptography (*e.g.* [GL03, Kal05, ACP09, BPV12]).

Smooth Projective Hashing System: A Smooth Projective Hash Function over a language $\mathcal{L} \subset X$, onto a set $\mathbb{G}$, is defined by five algorithms ($\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}$):

- $\mathsf{Setup}(1^{\mathfrak{K}})$ where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme, and the description of an $\mathcal{NP}$ language $\mathcal{L}$;
- $\mathsf{HashKG}(\mathcal{L}, \mathsf{param})$, outputs a hashing key hk for the language $\mathcal{L}$;
- $\mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W)$, derives the projection key hp, possibly depending on the word W [GL03, ACP09] thanks to the hashing key hk.
- $\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W)$, outputs a hash value $v \in \mathbb{G}$, thanks to the hashing key hk, and W
- $\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W, w)$, outputs the hash value $v' \in \mathbb{G}$, thanks to the projection key hp and the witness w that $W \in \mathcal{L}$.

In the following, we consider $\mathcal{L}$ as a hard-partitioned subset of X, *i.e.* it is computationally hard to distinguish a random element in $\mathcal{L}$ from a random element in $X \setminus \mathcal{L}$. A Smooth Projective Hash Function SPHF should satisfy the following properties:

- *Correctness*: Let $W \in \mathcal{L}$ and w a witness of this membership. For all hashing keys hk and associated projection keys hp we have $\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W) = \mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W, w)$.
- *Smoothness*: For all $W \in X \setminus \mathcal{L}$ the following distributions are statistically indistinguishable:

$$\mathit{\Delta}_0 = \left\{ (\mathcal{L}, \mathsf{param}, W, \mathsf{hp}, v) \;\middle|\; \begin{array}{l} \mathsf{param} = \mathsf{Setup}(1^{\mathfrak{K}}), \\ \mathsf{hk} = \mathsf{HashKG}(\mathcal{L}, \mathsf{param}), \\ \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W), \\ v = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W) \end{array} \right\}$$

$$\mathit{\Delta}_1 = \left\{ (\mathcal{L}, \mathsf{param}, W, \mathsf{hp}, v) \;\middle|\; \begin{array}{l} \mathsf{param} = \mathsf{Setup}(1^{\mathfrak{K}}), \\ \mathsf{hk} = \mathsf{HashKG}(\mathcal{L}, \mathsf{param}), \\ \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W), \\ v \overset{\$}{\leftarrow} \mathbb{G} \end{array} \right\}.$$

– *Pseudo-Randomness*: If $W \in \mathcal{L}$, then without a witness of membership the two previous distributions should remain computationally indistinguishable.

Classical Instantiations. For our needs, we consider discrete-logarithm based encryption schemes and related smooth projective hash functions. The underlying setting is a group $\mathbb{G}$ (denoted multiplicatively) of prime order p and we denote g a random generator of $\mathbb{G} = \langle g \rangle$. The security of our constructions will rely on the standard Decisional Diffie Hellman problems in $\mathbb{G}$:
*Decisional Diffie Hellman (*DDH*)* [Bon98]: The Decisional Diffie-Hellman hypothesis states that in a group $(p, \mathbb{G}, g)$ (written in multiplicative notation), given (g^μ, g^ν, g^ψ) for unknown $\mu, \nu \overset{\$}{\leftarrow} \mathbb{Z}_p$, it is hard to decide whether $\psi = \mu\nu$.

ElGamal encryption [ElG84] is defined by the following four algorithms:
- $\mathsf{Setup}(1^{\mathfrak{K}})$: The scheme needs a multiplicative group $(p, \mathbb{G}, g)$. The global parameters param consist of these elements $(p, \mathbb{G}, g)$.
- $\mathsf{KeyGen}(\mathsf{param})$: Chooses one random scalar $\alpha \overset{\$}{\leftarrow} \mathbb{Z}_p$, which define the secret key $\mathsf{dk} = \alpha$, and the public key $\mathsf{pk} = h = g^\alpha$.
- $\mathsf{Encrypt}(\mathsf{pk} = h, M; r)$: For a message $M \in \mathbb{G}$ and a random scalar $r \overset{\$}{\leftarrow} \mathbb{Z}_p$, computes the ciphertext as $C = \big(c_1 = h^r M, c_2 = g^r\big)$.
- $\mathsf{Decrypt}(\mathsf{dk} = \alpha, C = (c_1, c_2))$: One computes $M = c_1/(c_2^\alpha)$.

As shown by Boneh [Bon98], this scheme is IND-CPA under the hardness of DDH.

Cramer-Shoup encryption scheme [CS98] is an IND-CCA version of the ElGamal Encryption.
- $\mathsf{Setup}(1^{\mathfrak{K}})$ generates a group $\mathbb{G}$ of order p, with a generator g
- $\mathsf{KeyGen}(\mathsf{param})$ generates $(g_1, g_2) \overset{\$}{\leftarrow} \mathbb{G}^2$, $\mathsf{dk} = (x_1, x_2, y_1, y_2, z) \overset{\$}{\leftarrow} \mathbb{Z}_p^5$, and sets, $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$. It also chooses a Collision-Resistant hash function $\mathfrak{H}_K$ in a hash family $\mathcal{H}$ (or simply a Universal One-Way Hash Function). The encryption key is $\mathsf{ek} = (g_1, g_2, c, d, h, \mathfrak{H}_K)$.
- $\mathsf{Encrypt}(\mathsf{ek}, M; r)$, for a message $M \in \mathbb{G}$ and a random scalar $r \in \mathbb{Z}_p$, the ciphertext is $C = (\mathbf{u} = (g_1^r, g_2^r), e = M \cdot h^r, v = (cd^\xi)^r)$, where v is computed afterwards with $\xi = \mathfrak{H}_K(\mathbf{u}, e)$.
- $\mathsf{Decrypt}(\ell, \mathsf{dk}, C)$: one first computes $\xi = \mathfrak{H}_K(\mathbf{u}, e)$ and checks whether $u_1^{x_1 + \xi y_1} \cdot u_2^{x_2 + \xi y_2} \overset{?}{=} v$. If the equality holds, one computes $M = e/(u_1^z)$ and outputs M. Otherwise, one outputs $\perp$.

The security of the scheme is proven under the DDH assumption and the fact the hash function used is a Universal One-Way Hash Function (see [CS98]for details).

3 Security Model

Distributed PAKE. In a distributed PAKE system, we consider as usual a client (owning a password) willing to interact with a gateway, such as a website. The difference compared to a non-distributed system is that the gateway itself interacts with two servers, and none of the three owns enough information to be able to recover the passwords of the clients on its own[2]. Such a scheme is correct if the interaction between a client with a correct password and the gateway succeeds. An honest execution of a distributed PAKE protocol should result in the client holding a session key K_{U} and the gateway holding a session key $K_{\mathsf{G}} = K_{\mathsf{U}}$.

We propose in this paper two settings that describe well this situation. In a first setting, we consider that the passwords of the clients are shared information-theoretically between the servers, such as $\pi = \pi_1 + \pi_2$ (if the password π belongs to an appropriate group) or with the help of any secret sharing protocol. At the beginning of each time period, the shares are updated, in a probabilistic way, using a public function `Refresh`, depending on the sharing protocol used.

In a second setting, we consider that the gateway owns a database of encrypted passwords (which can be considered public), and the servers each own a share of the corresponding private keys (obtained by a secret sharing protocol). Again, at the beginning of each time period, the shares are updated, in a probabilistic way, using a public function `Refresh`, depending on the sharing protocol used.

Since the security of our schemes is not analyzed in the universal composability framework (contrary to the recent paper [CEN15]), the `Refresh` procedure can be handled easily using classical techniques from computational proactive secret sharing (see [OY91, HJKY95] for instance).

Security Model. We consider the classical model [BPR00] for authenticated key-exchange, adapted to the two-server setting by [ACFP05, KMTG12]. In the latter model, the authors assume that every client in the system shares its password with exactly two servers. We loosen this requirement here, depending on the setting considered, as described above. We refer the interested reader to these articles for the details and we give the high-level ideas in the Appendix A.

4 Our Simple Protocol

In this first setting, we consider a client U owning a password π and willing to interact with a gateway G. The gateway interacts with two servers S_1 (owning π_1) and S_2 (owning π_2), such that $\pi = \pi_1 + \pi_2$. It should be noted that only the client's password is assumed to be small and human-memorable. The two "passwords" owned by the servers can be arbitrarily big. The aim of the protocol is to establish a shared session key between the client and the gateway.

A simple solution to this problem consists in considering some sort of three-party PAKE, in which the client implicitly checks (using an SPHF) whether its password is the sum of the shares owned by the two servers, and the servers implicitly check (also using an SPHF) whether their share is the difference of the password owned by the client and the share owned by the other server. For sake of simplicity, we denote the client U as S_0 and its password π as π_0.

4.1 Building Blocks

Cramer-Shoup encryption and SPHF. We consider Cramer-Shoup encryption as described in Section 2. The public key is denoted by $\mathsf{ek} = (g_1, g_2, c, d, h, \mathfrak{H}_K)$ and the private key by $\mathsf{dk} = (x_1, x_2, y_1, y_2, z) \xleftarrow{\$} \mathbb{Z}_p^5$. The public parameters $(\mathbb{G}, p, g, \mathsf{ek})$ are given as a common reference string.

We denote the ciphertext of a message $M \in \mathbb{G}$ with the scalar $r \in \mathbb{Z}_p$ by $\mathcal{C} = \mathsf{CS}_{\mathsf{ek}}(M; r) = (u_1, u_2, e, v)$, with $v = cd^{\xi}$ and $\xi = \mathfrak{H}_K(u_1, u_2, e)$.

We use the SPHF described in [BBC$^+$13] for the language of the valid ciphertexts of M under the public key ek. Its main advantage is that it can be computed without using the associated ciphertext, and in particular before having seen it. This allows all the participants to send their ciphertext and their projected keys in only one flow. The classical use of this SPHF is as follows: user U (owning a message M)

[2] Note that the gateway can be merged with one server.

and V (owning a message M') are supposed to share a common message, so that $M = M'$. User U wants to implicitly check this equality. To this aim, user V sends an encryption $\mathcal{C}$ of M' under randomness r. In order for U to implicitly check that $\mathcal{C}$ is a valid encryption of M, it chooses a hash key hk and computes and sends a projection key hp to V. If $M = M'$ and if the encryption was computed correctly, then the hash value H computed by U using the private value hk is the same as the projected hash value H' computed by V using the public value hp and its private witness r. The SPHF is described by the following algorithms.

$$\begin{aligned}
\mathsf{Setup}(1^{\mathfrak{K}})\colon\ & \mathsf{param} = (\mathsf{ek}, M) \\
& \mathcal{L} = \{\mathcal{C} = (u_1, u_2, e, v) \in \mathbb{G}^4 \mid \exists r \in \mathbb{Z}_p \text{ such that} \\
& \qquad\qquad \mathcal{C} = \mathsf{CS}_{\mathsf{ek}}(M; r)\} \\
\mathsf{HashKG}(\mathcal{L}, (\mathsf{ek}, M))\colon\ & \mathsf{hk} = (\eta, \gamma, \theta, \lambda, \kappa) \xleftarrow{\$} \mathbb{Z}_p^5 \\
\mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, (\mathsf{ek}, M)))\colon\ & \mathsf{hp} = (\mathsf{hp}_1 = g_1^{\eta} g_2^{\theta} h^{\lambda} c^{\kappa}, \mathsf{hp}_2 = g_1^{\gamma} d^{\kappa}) \in \mathbb{G}^2 \\
\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, (\mathsf{ek}, M)), \mathcal{C})\colon\ & H = \mathsf{Hash}(\mathsf{hk}, (\mathsf{ek}, M), \mathcal{C}) = u_1^{(\eta+\xi\gamma)} u_2^{\theta} (e/M)^{\lambda} v^{\kappa} \\
\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, (\mathsf{ek}, M')), \mathcal{C}, r)\colon\ & H' = (\mathsf{hp}_1 \mathsf{hp}_2^{\xi})^r
\end{aligned}$$

It has been known to be correct, smooth and pseudo-random since [BBC+13].

Main Idea of the Construction. In our setting, we denote by $\mathsf{pw}_b = g^{\pi_b}$. The main idea of the protocol is depicted on Figure 1. For sake of readability, the participants which have a real role in the computations are directly linked by arrows in the picture, but one should keep in mind that all the participants (U, S_1 and S_2) only communicate with G, which then broadcasts all the messages.

In a classical SPHF-based two-party key-exchange between U and G, the client and the gateway would compute a Cramer-Shoup encryption of their password: $\mathcal{C}_0 = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_0; r_0)$ and $\mathcal{C}_{\mathsf{G}} = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_{\mathsf{G}}; r_{\mathsf{G}})$. The gateway would then send a projection key $\mathsf{hp}_{\mathsf{G},0}$ in order to implicitly check via an SPHF whether $\mathcal{C}_0$ is a valid Cramer-Shoup encryption of pw_{G}, and the client would send a projection key $\mathsf{hp}_{0,\mathsf{G}}$ in order to implicitly check via an SPHF whether $\mathcal{C}_{\mathsf{G}}$ is a valid Cramer-Shoup encryption of pw_0.

Here, since S_0 owns $\mathsf{pw}_0 = \mathsf{pw}_1 \cdot \mathsf{pw}_2$, so that the players do not share the same password, we consider an SPHF between each pair of players $(\mathcal{S}_i, \mathcal{S}_j)$, in which player $\mathcal{S}_i$ computes the ciphertext $\mathcal{C}_i = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_i; r_i)$, the keys $\mathsf{hk}_{i,j}$ and $\mathsf{hp}_{i,j}$ and sends $(\mathcal{C}_i, \mathsf{hp}_{i,j})$ to $\mathcal{S}_j$. It also computes the hash value $H_{i,j} = \mathsf{Hash}(\mathsf{hk}_{i,j}, (\mathsf{ek}, \mathsf{pw}_i), \mathcal{C}_j)$ and projected hash value $H'_{j,i} = \mathsf{ProjHash}(\mathsf{hp}_{j,i}, (\mathsf{ek}, M_i), \mathcal{C}_i, r_i)$. Formally, for each pair of users $(\mathcal{S}_i, \mathcal{S}_j)$, the language checked on $\mathcal{S}_j$ by $\mathcal{S}_i$ is defined as follows: $\mathcal{C}_j \in \mathcal{L}_{i,j} = \{\mathcal{C} = (u_1, u_2, e, v) \in \mathbb{G}^4 \mid \exists r \in \mathbb{Z}_p \text{ such that } \mathcal{C} = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_j; r)\}$ but it cannot be checked directly by a unique SPHF since the passwords are different (and thus $\mathcal{S}_i$ does not know pw_j). Rather, we combine in the protocol the six SPHF described to globally ensure the correctness (each one remaining smooth and pseudo-random), as described in the next part. The correctness of the SPHF for the pair $(\mathcal{S}_i, \mathcal{S}_j)$ implies that if everything was computed honestly, then one gets the equalities $H_{i,j}(\mathsf{pw}_i/\mathsf{pw}_j)^{\lambda_i} = H'_{i,j}$ and $H_{j,i}(\mathsf{pw}_j/\mathsf{pw}_i)^{\lambda_j} = H'_{j,i}$.

4.2 Login procedure

- Each participant S_b picks r_b at random and computes a Cramer-Shoup encryption of its password $\mathcal{C}_b = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_b; r_b)$, with $v_b = cd^{\xi_b}$.
  It also chooses, for $i \in \{0, 1, 2\} \setminus \{b\}$, a random hash key $\mathsf{hk}_{b,i} = (\eta_{b,i}, \gamma_{b,i}, \theta_{b,i}, \lambda_b, \kappa_{b,i})$ and sets $\mathsf{hp}_{b,i} = (\mathsf{hp}_{b,i;1}, \mathsf{hp}_{b,i;2}) = ({g_1}^{\eta_{b,i}} {g_2}^{\theta_{b,i}} h^{\lambda_b} c^{\kappa_{b,i}}, {g_1}^{\gamma_{b,i}} d^{\kappa_{b,i}})$ as the projection key intended to the participant S_i.
  It sends $(\mathcal{C}_b, (\mathsf{hp}_{b,i})_{i \in \{0,1,2\} \setminus \{b\}})$ to the gateway G, which broadcasts these values to the other participants.
- After receiving the first flow from the servers, the client computes $H'_{i,0} = {\mathsf{hp}_{i,0;1}}^{r_0} {\mathsf{hp}_{i,0:2}}^{\xi_0 r_0}$ for $i \in \{1, 2\}$. It also computes $H_0 = H_{0,1} \cdot H_{0,2} \cdot {\mathsf{pw}_0}^{\lambda_0}$, and sets its session key K_{U} as $K_{\mathsf{U}} = K_0 = H'_{1,0} \cdot H'_{2,0} \cdot H_0$.

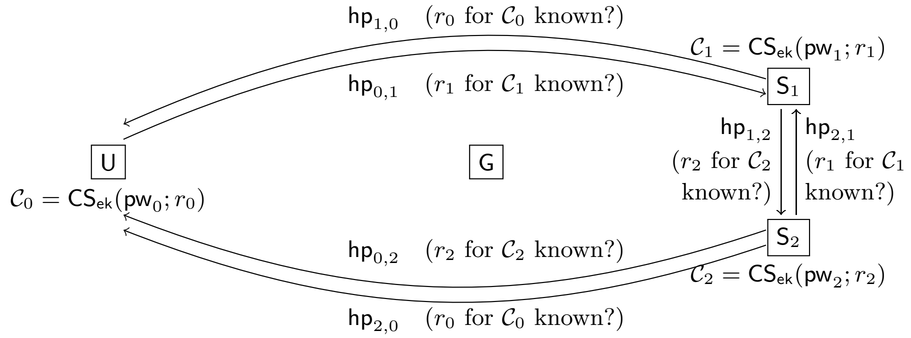
$\mathsf{hp}_{1,0}$ (r_0 for $\mathcal{C}_0$ known?)
$\mathcal{C}_1 = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_1; r_1)$
$\mathsf{hp}_{0,1}$ (r_1 for $\mathcal{C}_1$ known?)
S_1
$\mathsf{hp}_{1,2}$ (r_2 for $\mathcal{C}_2$ known?)
$\mathsf{hp}_{2,1}$ (r_1 for $\mathcal{C}_1$ known?)
U
G
$\mathcal{C}_0 = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_0; r_0)$
S_2
$\mathsf{hp}_{0,2}$ (r_2 for $\mathcal{C}_2$ known?)
$\mathcal{C}_2 = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_2; r_2)$
$\mathsf{hp}_{2,0}$ (r_0 for $\mathcal{C}_0$ known?)

Fig. 1. Main idea of the construction

After receiving the first flow from the other server and the client, the server S_b computes, for $i \in \{0, 3-b\}$, $H'_{i,b} = \mathsf{hp}_{i,b;1}{}^{r_b}\mathsf{hp}_{i,b;2}{}^{\xi_b r_b}$. It also computes $H_b = H_{b,0}/(H_{b,3-b} \cdot \mathsf{pw}_b^{\lambda_b})$, and sets its partial key K_b as $K_b = H'_{0,b} \cdot H'_{3-b,b} \cdot H_b$. It privately sends this value K_b to the gateway G.

- The gateway finally sets $K_\mathsf{G} = K_1 \cdot K_2$.

Correctness. Recall that $H'_{i,j} = H_{i,j}(\mathsf{pw}_i/\mathsf{pw}_j)^{\lambda_i}$ for all pairs $(i,j) \in \{0,1,2\}^2$, so that the session key of the gateway is equal to $K_\mathsf{G} = K_1 \cdot K_2$, *i.e.*

$K_\mathsf{G} = \frac{H'_{0,1}H'_{2,1}H_{1,0}}{H_{1,2}\mathsf{pw}_1^{\lambda_1}} \frac{H'_{0,2}H'_{1,2}H_{2,0}}{H_{2,1}\mathsf{pw}_2^{\lambda_2}} = H_{0,1}H_{1,0}H_{0,2}H_{2,0}\left(\frac{1}{\mathsf{pw}_2}\right)^{\lambda_1}\left(\frac{1}{\mathsf{pw}_1}\right)^{\lambda_2}\left(\frac{(\mathsf{pw}_0)^2}{\mathsf{pw}_1\mathsf{pw}_2}\right)^{\lambda_0}$

while the session key of the client is $K_\mathsf{U} = H'_{1,0} \cdot H'_{2,0} \cdot H_0$, *i.e.*

$K_\mathsf{U} = H'_{1,0} \cdot H'_{2,0} \cdot H_{0,1} \cdot H_{0,2} \cdot \mathsf{pw}_0{}^{\lambda_0} = H_{0,1}H_{1,0}H_{0,2}H_{2,0}\left(\frac{\mathsf{pw}_1}{\mathsf{pw}_0}\right)^{\lambda_1}\left(\frac{\mathsf{pw}_2}{\mathsf{pw}_0}\right)^{\lambda_2}(\mathsf{pw}_0)^{\lambda_0}$

and these two values are equal as soon as $\mathsf{pw}_0 = \mathsf{pw}_1 \cdot \mathsf{pw}_2$.

Complexity. The following table sums up the number of group elements needed for each participant. It should be noted that in case one of the servers is the gateway, its communication costs are reduced.

	Ciphertext (Broadcast)	Projection Keys (Overall)	To the Gateway
Client	4	4	0
Server (each)	4	4	1

Proof of Security. The proof follows the spirit of the proofs given in [KV11, BBC$^+$13]. For lack of space, a sketch is given in the Appendix B.

5 Our Efficient Protocol

In this second setting, we consider again a client U owning a password π and willing to interact with a gateway G. The gateway owns a public database of encrypted passwords, and it interacts with two servers S_1 and S_2, each owning a share of the secret key of the encryption scheme. The aim of the protocol is to establish a shared session key between the client and the gateway.

The idea is similar to the protocol described in the former section, except that only the client needs to compute a ciphertext, the other ciphertext being publicly available from the database. The participants implicitly check (using several SPHF) that the message encrypted in the ciphertext of the client is the same as the message encrypted in the database (using the secret key shared upon the servers).

5.1 Building Blocks

Cramer-Shoup encryption and SPHF. We consider Cramer-Shoup encryption as described in Section 2. The public key is denoted by $\mathsf{ek} = (g_1, g_2, c, d, h, \mathfrak{H}_K)$ and the private key by $\mathsf{dk} = (x_1, x_2, y_1, y_2, z) \overset{\$}{\leftarrow} \mathbb{Z}_p^5$. The public parameters $(\mathbb{G}, p, g, \mathsf{ek})$ are given as a common reference string.

We denote the ciphertext of a message $M \in \mathbb{G}$ with the scalar $r \in \mathbb{Z}_p$ by $\mathcal{C} = \mathsf{CS}_{\mathsf{ek}}(M; r) = (u_1, u_2, e, v)$, with $v = cd^{\xi}$ and $\xi = \mathfrak{H}_K(u_1, u_2, e)$.

We use here the simpler SPHF described in [GL03] for the language of the valid ciphertexts of M under the public key ek, in which the participants need to see the ciphertext before being able to compute their projection keys. Since only the servers will be required to compute such projection keys and since they are not required to compute a ciphertext anymore, all the participants will be able to do all their computation in only one round as in the former protocol. The SPHF is described by the following algorithms.

$$\begin{aligned}
\mathsf{Setup}(1^{\mathfrak{K}})\colon\ & \mathsf{param} = (\mathsf{ek}, M) \\
& \quad \mathcal{L} = \{\mathcal{C} = (u_1, u_2, e, v) \in \mathbb{G}^4 \mid \exists r \in \mathbb{Z}_p \text{ such that} \\
& \qquad\qquad \mathcal{C} = \mathsf{CS}_{\mathsf{ek}}(M; r)\} \\
\mathsf{HashKG}(\mathcal{L}, (\mathsf{ek}, M))\colon\ & \mathsf{hk} = (\eta, \theta, \lambda, \kappa) \overset{\$}{\leftarrow} \mathbb{Z}_p^4 \\
\mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, (\mathsf{ek}, M)), \mathcal{C})\colon\ & \mathsf{hp} = g_1^{\eta} g_2^{\theta} h^{\lambda} (cd^{\xi})^{\kappa} \in \mathbb{G} \\
\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, (\mathsf{ek}, M)), \mathcal{C})\colon\ & H = u_1^{\eta} u_2^{\theta} (e/M)^{\lambda} v^{\kappa} \\
\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, (\mathsf{ek}, M')), \mathcal{C}, r)\colon\ & H' = \mathsf{hp}^r
\end{aligned}$$

It has been known to be correct, smooth and pseudo-random since [GL03].

El Gamal encryption and SPHF. We consider El Gamal encryption as described in Section 2. The public key is denoted by $\mathsf{pk} = h = g^{\alpha}$ and the private key by $\mathsf{sk} = \alpha \overset{\$}{\leftarrow} \mathbb{Z}_p$. The public parameters $(\mathbb{G}, p, g, \mathsf{pk})$ are given as a common reference string. We denote the ciphertext of a message $M \in \mathbb{G}$ with the scalar $r \in \mathbb{Z}_p$ by $\mathcal{C} = \mathsf{EG}_{\mathsf{pk}}(M; r) = (e, u) = (h^r M, g^r)$.

The regular SPHF used throughout the literature [CS02] on the language $\{\mathcal{C} = (e, u) \in \mathbb{G}^2 \mid \exists r \in \mathbb{Z}_p \text{ such that } \mathcal{C} = \mathsf{EG}_{\mathsf{pk}}(M; r)\}$ of the valid encryptions of M, with r being the witness of the word $\mathcal{C}$, usually allows a server to check whether the ciphertext was honestly computed on the value M by a client, by generating a pair of keys $(\mathsf{hk}, \mathsf{hp})$ and sending hp to the client.

Here, on the contrary, we now want a client to implicitly check that a server knows the decryption key of the encryption scheme. This means that the client computes both the ciphertext (or the ciphertext is publicly available in a database, as here) and the pair of keys $(\mathsf{hk}, \mathsf{hp})$ and sends the ciphertext as well as hp to the server, which now uses the decryption key as a witness. This implies the following modifications to the algorithms of the SPHF:

$$\begin{aligned}
\mathsf{Setup}(1^{\mathfrak{K}})\colon\ & \mathsf{param} = (\mathsf{pk}, M) \\
& \quad \mathcal{L} = \{\mathcal{C} = (e, u) \in \mathbb{G}^2 \mid \exists \alpha \in \mathbb{Z}_p \text{ such that} \\
& \qquad\qquad h = g^{\alpha} \text{ and } e/u^{\alpha} = M\} \\
\mathsf{HashKG}(\mathcal{L}, (\mathsf{pk}, M))\colon\ & \mathsf{hk} = (\lambda, \mu) \overset{\$}{\leftarrow} \mathbb{Z}_p^2 \\
\mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, (\mathsf{pk}, M)), \mathcal{C})\colon\ & \mathsf{hp} = u^{\lambda} g^{\mu} \in \mathbb{G} \\
\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, (\mathsf{pk}, M)), \mathcal{C})\colon\ & H = h^{\mu} (e/M)^{\lambda} \\
\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, (\mathsf{pk}, M')), \mathcal{C}, \alpha)\colon\ & H' = \mathsf{hp}^{\alpha}
\end{aligned}$$

and we show that this SPHF satisfies the usual properties:

- *Correctness*: if $\mathcal{C} \in \mathcal{L}$ with witness α, one directly gets $H = H'$;
- *Smoothness*: assume $\mathcal{C} \notin \mathcal{L}$. It can then be parsed as $\mathcal{C} = (e, u)$ with $u = g^r$ and $e = h^r M' = g^{\alpha r} M g^{\delta_M}$ (with $M' \neq M$ and thus $\delta_M \neq 0$), so that $\mathsf{hp} = u^{\lambda} g^{\mu} = g^{r\lambda + \mu}$ and $H = h^{\mu}(e/M)^{\lambda} = g^{\alpha\mu} g^{(\alpha r + \delta_M)\lambda}$.

The smoothness is then easy to see by considering the following equality of matrices:

$$\begin{pmatrix} \log_g(\mathsf{hp}) \\ \log_g(H) \end{pmatrix} = \begin{pmatrix} r & 1 \\ \alpha r + \delta_M & \alpha \end{pmatrix} \cdot \begin{pmatrix} \lambda \\ \mu \end{pmatrix}$$

which shows that as soon as the word is not a valid encryption of M, and so δ_M is not equal to 0, the Hash value H is not in the span of the projection key hp so that the two distributions described in Section 2 are indistinguishable.

- *Pseudo-Randomness.* Since El Gamal encryption is $\mathsf{IND\text{-}CPA}$, it is impossible to distinguish between a real encryption and a random value without knowing the decryption key. Since the decryption key is the witness of the SPHF, without knowing the witness, the two distributions remain indistinguishable.

Main Idea of the Construction. Again, we denote the client U as S_0 and its password π as π_0. In our setting, we denote by $\mathsf{pw}_k = g^{\pi_k}$ for all k. The database contains El Gamal encryptions of each ciphertext $\mathsf{pw}_{\mathsf{U}_i}$, under randomness s_{U_i}: $\mathcal{C}^{db}_{\mathsf{U}_i} = \mathsf{EG}_{\mathsf{pk}}(\mathsf{pw}_{\mathsf{U}_i}; s_{\mathsf{U}_i}) = (h^{s_{\mathsf{U}_i}}\mathsf{pw}_{\mathsf{U}_i}, g^{s_{\mathsf{U}_i}})$, so that here, $\mathcal{C}^{db}_{\mathsf{U}} = \mathsf{EG}_{\mathsf{pk}}(\mathsf{pw}_{\mathsf{U}}; s_{\mathsf{U}}) = (h^{s_{\mathsf{U}}}\mathsf{pw}_{\mathsf{U}}, g^{s_{\mathsf{U}}})$. The client computes a Cramer-Shoup encryption of its password: $\mathcal{C}_0 = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_0; r_0) = (u_1, u_2, e, v)$ with $v = cd^{\xi}$. The execution of the protocol should succeed if these encryptions are correct and $\mathsf{pw}_0 = \mathsf{pw}_{\mathsf{U}}$. Recall that the server $\mathcal{S}_i$ knows α_i such that $\alpha = \alpha_1 + \alpha_2$ is the decryption key of the El Gamal encryption.

The main idea is depicted on Figure 2. For sake of readability, the participants which have a real role in the computations are directly linked by arrows in the picture, but one should keep in mind that all the participants (U, S_1 and S_2) only communicate with G, which then broadcasts all the messages.

In a classical SPHF-based two-party key-exchange between U and G, the gateway would check if $\mathcal{C}_0$ is a valid Cramer-Shoup encryption of pw_{U}. Since here the password pw_{U} is unknown to the servers S_1 and S_2, this is done in our setting by two SPHF, using $\mathsf{hp}_1^{\mathrm{CS}}$ (sent by S_1) and $\mathsf{hp}_2^{\mathrm{CS}}$ (sent by S_2), where the servers use the first term of the public encryption $\mathcal{C}^{\mathrm{DB}}_{\mathsf{U}}$ ($h^{s_{\mathsf{U}}}\mathsf{pw}_{\mathsf{U}}$) in order to cancel the unknown pw_{U}.

In a classical SPHF-based two-party key-exchange between U and G, the client would also check whether $\mathcal{C}^{\mathrm{DB}}_{\mathsf{U}}$ is a valid El Gamal encryption of its password pw_0, *i.e.* whether the gateway knows a witness for its ciphertext $\mathcal{C}^{\mathrm{DB}}_{\mathsf{U}}$ (s_{U} in the usual constructions, α here). Since α is unknown to the gateway, this is done in our setting by the combination of three SPHF, using $\mathsf{hp}_0^{\mathrm{EG}}$ (sent by the client), $\mathsf{hp}_1^{\mathrm{EG}}$ (sent by S_1) and $\mathsf{hp}_2^{\mathrm{EG}}$ (sent by S_2). These three SPHF allow the client and the servers to implicitly check that the servers know α_1 and α_2 such that $\mathcal{C}^{\mathrm{DB}}_{\mathsf{U}}$ can be decrypted (using the decryption key $\alpha = \alpha_1 + \alpha_2$) to the same password pw_0 than the one encrypted in $\mathcal{C}_0$ sent by the client. Formally, the languages checked are as follows:

- by the client:
  $\mathcal{C}^{\mathrm{DB}}_{\mathsf{U}} \in \mathcal{L}_0 = \{\mathcal{C} = (e, u) \in \mathbb{G}^2 \mid \exists \alpha \in \mathbb{Z}_p \text{ such that } h = g^{\alpha} \text{ and } e/u^{\alpha} = \mathsf{pw}_0\}$
- by server $\mathcal{S}_i$ (with respect to the client $\mathcal{S}_0$ and server $\mathcal{S}_j$):
  $\mathcal{C}_0 \in \mathcal{L}_{i,0} = \{\mathcal{C} = (u_1, u_2, e, v) \in \mathbb{G}^4 \mid \exists r \in \mathbb{Z}_p \text{ such that } \mathcal{C} = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_{\mathsf{U}}; r)$ and $\mathcal{C}^{\mathrm{DB}}_{\mathsf{U}} \in \mathcal{L}_{i,j} = \{\mathcal{C} = (e, u) \in \mathbb{G}^2 \mid \exists \alpha_j \in \mathbb{Z}_p \text{ such that } h = g^{\alpha_i + \alpha_j}\}$ and $e/u^{\alpha_i+\alpha_j} = \mathsf{pw}_{\mathsf{U}}\}$

but they cannot be checked directly by a unique SPHF since the value pw_{U} appearing in the languages is unknown to the verifier $\mathcal{S}_i$. Rather, the server $\mathcal{S}_i$ will use the first term of the public encryption $\mathcal{C}^{\mathrm{DB}}_{\mathsf{U}}$ ($h^{s_{\mathsf{U}}}\mathsf{pw}_{\mathsf{U}}$) in order to cancel this unknown pw_{U}. To achieve this goal, we combine the five SPHF described to globally ensure the correctness (each one remaining smooth and pseudo-randomness), as described in the next part.

5.2 Login procedure

- The client $\mathsf{U} = \mathsf{S}_0$ chooses r_0 at random and computes a Cramer-Shoup encryption of its password $\mathcal{C}_0 = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_0; r_0) = (u_1, u_2, e, v)$ with $v = cd^{\xi}$.
  It also chooses a random (El Gamal) hash key $\mathsf{hk}_0^{\mathrm{EG}} = (\lambda_0, \mu_0)$ and computes the corresponding projected key on the ciphertext $\mathcal{C}^{\mathrm{DB}}_{\mathsf{U}} = \mathsf{EG}_{\mathsf{pk}}(\mathsf{pw}_{\mathsf{U}}; s_{\mathsf{U}}) = (h^{s_{\mathsf{U}}}\mathsf{pw}_{\mathsf{U}}, g^{s_{\mathsf{U}}})$ contained in the database: $\mathsf{hp}_0^{\mathrm{EG}} = g^{s_{\mathsf{U}}\lambda_0} g^{\mu_0}$.

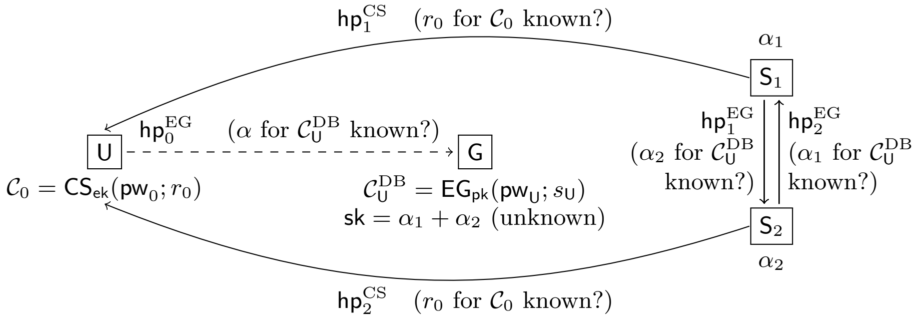
$\mathsf{hp}_1^{\mathrm{CS}}$ (r_0 for $\mathcal{C}_0$ known?)
α_1
S_1
$\mathsf{hp}_0^{\mathrm{EG}}$ (α for $\mathcal{C}_\mathsf{U}^{\mathrm{DB}}$ known?)
U
G
$\mathsf{hp}_1^{\mathrm{EG}}$ (α_2 for $\mathcal{C}_\mathsf{U}^{\mathrm{DB}}$ known?)
$\mathsf{hp}_2^{\mathrm{EG}}$ (α_1 for $\mathcal{C}_\mathsf{U}^{\mathrm{DB}}$ known?)
$\mathcal{C}_0 = \mathsf{CS}_{\mathsf{ek}}(\mathsf{pw}_0; r_0)$
$\mathcal{C}_\mathsf{U}^{\mathrm{DB}} = \mathsf{EG}_{\mathsf{pk}}(\mathsf{pw}_\mathsf{U}; s_\mathsf{U})$
$\mathsf{sk} = \alpha_1 + \alpha_2$ (unknown)
S_2
α_2
$\mathsf{hp}_2^{\mathrm{CS}}$ (r_0 for $\mathcal{C}_0$ known?)

Fig. 2. Main idea of the construction

It sends $(\mathcal{C}_0, \mathsf{hp}_0^{\mathrm{EG}})$ to the gateway, which broadcasts these values to the servers.

- After receiving this flow from the client, the servers S_1 and S_2 also choose a random (El Gamal) hash key $\mathsf{hk}_b^{\mathrm{EG}} = (\lambda_b, \mu_b)$ and compute the corresponding projected key on the ciphertext $\mathcal{C}_\mathsf{U}^{\mathrm{DB}}$ contained in the database: $\mathsf{hp}_b^{\mathrm{EG}} = g^{s_\mathsf{U}\lambda_b} g^{\mu_b}$.
  The servers S_1 and S_2 also choose a random (Cramer-Shoup) hash key $\mathsf{hk}_b^{\mathrm{CS}} = (\eta_b, \theta_b, \lambda_b, \kappa_b)$ (with the same value λ_b) and compute the corresponding projected key on the ciphertext $\mathcal{C}_0$ sent by the client: $\mathsf{hp}_b^{\mathrm{CS}} = ({g_1}^{\eta_b} {g_2}^{\theta_b} h^{\lambda_b} (cd^{\xi})^{\kappa_b})$.
  Each server sends $(\mathsf{hp}_b^{\mathrm{EG}}, \mathsf{hp}_b^{\mathrm{CS}})$ to the gateway G, which broadcasts these values to the other participants.
- After receiving the projected keys from the servers, the client computes, for $i \in \{1, 2\}$, $H'_{i,0} = (\mathsf{hp}_i^{\mathrm{CS}})^{r_0}$. It also computes $H_0 = (h^{s_\mathsf{U}} \mathsf{pw}_\mathsf{U}/\mathsf{pw}_0)^{\lambda_0} h^{\mu_0}$ by dividing the first term of the ciphertext $\mathcal{C}_\mathsf{U}^{\mathrm{DB}}$ contained in the database by its password pw_0. It finally sets its session key K_U as $K_\mathsf{U} = K_0 = H'_{1,0} \cdot H'_{2,0} \cdot H_0$.
  After receiving the first flow from the other server and the client, the server S_b computes, for $i \in \{0, 3-b\}$, $H'_{i,b} = (\mathsf{hp}_i^{\mathrm{EG}})^{\alpha_b}$.
  It also computes $H_{b,0} = (u_1)^{\eta_b}(u_2)^{\theta_b}[e/(h^{s_\mathsf{U}}\mathsf{pw}_\mathsf{U})]^{\lambda_b} v^{\kappa_b}$ and $H_b = H_{b,0} \cdot (\mathsf{hp}_b^{\mathrm{EG}})^{\alpha_b}/h^{\mu_b}$, and sets its partial key K_b as $K_b = H'_{0,b} \cdot H'_{3-b,b} \cdot H_b$. It privately sends this value K_b to the gateway G.
- The gateway finally sets $K_\mathsf{G} = K_1 \cdot K_2$.

Correctness. Due to the correctness of the Cramer-Shoup SPHF, we have $H'_{b,0} = H_{b,0}\left(\frac{h^{s_\mathsf{U}}\mathsf{pw}_\mathsf{U}}{\mathsf{pw}_0}\right)^{\lambda_b}$ for $b \in \{1, 2\}$. The session key of the gateway is thus equal to $K_\mathsf{G} = K_1 \cdot K_2 = (H'_{0,1} \cdot H'_{2,1} \cdot H_1) \cdot (H'_{0,2} \cdot H'_{1,2} \cdot H_2)$ where, after computation, $K_b = h^{(\lambda_0+\lambda_1+\lambda_2)\alpha_b} g^{(\mu_0+\mu_1+\mu_2)\alpha_b} H'_{b,0} (\mathsf{pw}_0/\mathsf{pw}_\mathsf{U})^{\lambda_b} h^{-s_\mathsf{U}\lambda_b} h^{-\mu_b}$.
If $\mathsf{pw}_0 = \mathsf{pw}_\mathsf{U}$, $h = g^{\alpha}$ and $\alpha = \alpha_1 + \alpha_2$, $K_\mathsf{G} = h^{s_\mathsf{U}\lambda_0} h^{\mu_0} H'_{1,0} H'_{2,0}$, which is equal to the session key of the client $K_\mathsf{U} = K_0 = H'_{1,0} \cdot H'_{2,0} \cdot H_0 = H'_{1,0} \cdot H'_{2,0} \cdot h^{s_\mathsf{U}\lambda_0} h^{\mu_0}$.

Complexity. The following table sums up the number of group elements needed for each participant. It should be noted that in case one of the servers is the gateway, its communication costs are reduced.

	Ciphertext (Broadcast)	Projection Keys (Overall)	To the Gateway
Client	4	2	0
Server (each)	0	2	1

Compared to [KMTG12], the communication complexity of our protocol is decreased by more than 50% (9 group elements instead of 20 group elements). For efficiency, as in [KMTG12], we count exponentiations only, and assume a multi-exponentiation with up to 5 bases can be computed at the cost of at most 1.5 exponentiations. The client performs the equivalent of 8 full exponentiations, while each server performs 7 exponentiations (instead of 15 and 13 respectively in [KMTG12]).

Proof of Security. The proof follows the idea of the former one. For lack of space, a sketch is given in the Appendix B.

6 Conclusion

We presented two constructions of distributed Password-Authenticated Key Exchange between a user and several servers. We focused on presenting them in a classical group setting (with only two servers). Very efficient implementations of our protocols can be readily obtained using standard cryptographic libraries and do not require pairings.

Our methods can be generalized to the setting where n servers share the (encryption of the) password. SPHF can further handle polynomials of variables and the use of secret sharing techniques *à la* Shamir, allows to share polynomials evaluation between n servers and to provide a threshold distributed PAKE such that security is ensured as long as less than a certain arbitrary threshold $t \in \{1, \ldots, n\}$ of servers are compromised (contrary to the protocol from [DG06] which requires an honest majority of the servers).

Smooth projective hashing is mostly used in a classical discrete-logarithm-based setting (or pairing-based setting) but constructions were also proposed for Paillier encryption [CS02] and for LWE encryption [KV09]. These SPHF would allow a readily adaptation of our techniques to other classical settings of cryptography.

Acknowledgements. This work was supported in part by the French ANR Project ANR-14-CE28-0003 EnBiD.

References

[ACFP05] Michel Abdalla, Olivier Chevassut, Pierre-Alain Fouque, and David Pointcheval. A simple threshold authenticated key exchange from short secrets. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 566–584. Springer, December 2005.

[ACP09] Michel Abdalla, Céline Chevalier, and David Pointcheval. Smooth projective hashing for conditionally extractable commitments. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 671–689. Springer, August 2009.

[BBC$^+$13] Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 449–475. Springer, August 2013.

[BJKS03] John G. Brainard, Ari Juels, Burt Kaliski, and Michael Szydlo. A new two-server approach for authentication with short secrets. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*, 2003.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.

[Bon98] Dan Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *LNCS*. Springer, 1998. Invited paper.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, May 2000.

[BPV12] Olivier Blazy, David Pointcheval, and Damien Vergnaud. Round-optimal privacy-preserving protocols with smooth projective hash functions. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 94–111. Springer, March 2012.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CEN15] Jan Camenisch, Robert R. Enderlein, and Gregory Neven. Two-server password-authenticated secret sharing UC-secure against transient corruptions. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 283–307. Springer, March / April 2015.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, August 1998.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, April / May 2002.

[DG03] Mario Di Raimondo and Rosario Gennaro. Provably secure threshold password-authenticated key exchange. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 507–523. Springer, May 2003.

[DG06] Mario Di Raimondo and Rosario Gennaro. Provably secure threshold password-authenticated key exchange. *J. Comput. Syst. Sci.*, 72(6):978–1001, 2006.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, August 1984.

[FK00] Warwick Ford and Burton S. Kaliski Jr. Server-assisted generation of a strong secret from a password. In *9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2000), 4-16 June 2000, Gaithersburg, MD, USA*, pages 176–180, 2000.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, May 2003. `http://eprint.iacr.org/2003/032.ps.gz`.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 339–352. Springer, August 1995.

[Jab01] David P. Jablon. Password authentication using multiple servers. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 344–360. Springer, April 2001.

[Kal05] Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 78–95. Springer, May 2005.

[KM14] Franziskus Kiefer and Mark Manulis. Distributed smooth projective hashing and its application to two-server password authenticated key exchange. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *ACNS 14*, volume 8479 of *LNCS*, pages 199–216. Springer, June 2014.

[KMTG05] Jonathan Katz, Philip D. MacKenzie, Gelareh Taban, and Virgil D. Gligor. Two-server password-only authenticated key exchange. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *ACNS 05*, volume 3531 of *LNCS*, pages 1–16. Springer, June 2005.

[KMTG12] Jonathan Katz, Philip D. MacKenzie, Gelareh Taban, and Virgil D. Gligor. Two-server password-only authenticated key exchange. *J. Comput. Syst. Sci.*, 78(2):651–669, 2012.

[KOY01] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 475–494. Springer, May 2001.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 636–652. Springer, December 2009.

[KV11] Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, March 2011.

[MSJ02] Philip D. MacKenzie, Thomas Shrimpton, and Markus Jakobsson. Threshold password-authenticated key exchange. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 385–400. Springer, August 2002.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *10th ACM PODC*, pages 51–59. ACM, August 1991.

[Poi12] David Pointcheval. Password-based authenticated key exchange (invited talk). In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 390–397. Springer, May 2012.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 433–444. Springer, August 1992.

[SK05] Michael Szydlo and Burton S. Kaliski Jr. Proofs for two-server password authentication. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 227–244. Springer, February 2005.

A Security Model

As in [KMTG12], we consider an adversarial model in which the attacker is active and can play on behalf of a user, or corrupt a server, or play on behalf of the gateway (to perform an online dictionary attack).

We assume the existence of a timestamp, only allowing the adversary to corrupt one server once during each time period, which means that the client interacts with at most one corrupted server. We also assume that the gateway is the only entity which directly communicates with the client: messages to and from the servers and the client are all transmitted by the gateway. Similarly, we assume there is no direct communication between the servers. For the sake of efficiency, the gateway may sometimes be merged with one of the servers. As in [ACFP05], we assume the existence of a private channel between the gateway and the servers, only used for the final computation of the session key of the gateway (we only need the secret communication to be possible from the servers to the gateway).

Each participant is assumed to be able to execute the protocol many times, possibly concurrently, which is modeled by allowing it to have an unlimited number of instances (used only once). Each instance maintains a local state during the execution of the protocol and is associated with several variables: its session identifier (*i.e.* its identity), its partner identifier (*i.e.* the identity of the participants with whom it believes to be interacting), its session key and two boolean variables useful to indicate whether the instance has terminated and/or accepted. Terminated means that the instance has finished sending and receiving messages according to the description of the protocol whereas in our case, accepted means that the instance believes it has established a session key with its partner.

The session identifier of an instance is defined as the ordered sequence of incoming and outgoing messages for this instance sent through the gateway. This is clear from the oracle queries made by the adversary. Instances of client and servers are called partners if session identifiers are equal (they share the same transcript of execution) and the identity of each is included into the partner identifier of the other. Since the gateway does not formally participate in the computation, an instance j of the servers S_1 and S_2 implicitly defines an instance j of the gateway.

The adversary is assumed to have complete control over the public communication between the participants and the gateway. This is formally modeled by letting him have access to the instances via the following oracles. Note that the adversary also receives the internal state of any corrupted participant. Furthermore, since we consider active corruptions, he also controls all messages sent by corrupted participants.

- $\mathsf{Send}(\mathsf{U}, i, m)$: this sends message m to the gateway, which then forwards it appropriately to the i^{th} instance of the client U, which behaves in response as described in the protocol. Its output is given to the adversary.
- $\mathsf{Send}(\mathsf{G}, j, m)$: this sends message m to the gateway, which then forwards it appropriately to the j^{th} instance of the servers S_1 and S_2, which behave in response as described in the protocol. Their output is given to the adversary.
- $\mathsf{Execute}(\mathsf{U}, i, \mathsf{G}, j)$: this oracle executes the protocol between the i^{th} instance of the client C and the j^{th} instance of the servers S_1 and S_2 (via the gateway) and outputs the transcript of the execution (*i.e.* all the public communication) to the adversary.
- $\mathsf{Reveal}(\mathsf{U}, i, \mathsf{G}, j)$: this outputs the session key hold by the i^{th} instance of the client U in an execution with the j^{th} instance of the servers S_1 and S_2 (assuming these participants are partners). This models possible leakage of the key for several reasons such as misuse, compromise of a computer, etc. Similarly, $\mathsf{Reveal}(\mathsf{G}, j, \mathsf{U}, i)$ outputs the session key hold by the gateway G in an execution between the i^{th} instance of the client U and the j^{th} instance of the servers S_1 and S_2 (assuming these participants are partners).
- $\mathsf{Test}(\mathsf{U}, i, \mathsf{G}, j)$: this is the query needed in order to properly define the security of the protocol. Assuming the i^{th} instance of the client U and the j^{th} instance of the servers S_1 and S_2 are partners, the oracle outputs $\perp$ if the session key of the i^{th} instance of the client C computed during an execution with the j^{th} instance of the servers has not yet been set. Otherwise, it generates a random bit b: if $b = 1$, the real session key is given to the adversary, whereas if $b = 0$, a random session key is given a random session key. The query $\mathsf{Test}(\mathsf{G}, j, \mathsf{U}, i)$ on the session key computed by the gateway G in an execution between the i^{th} instance of the client U and the j^{th} instance of the servers S_1 and S_2 is defined the same way. The adversary is allowed a single Test query at any time during the execution of the protocol.

The correctness of an execution of the protocol is defined as follows: if the i^{th} instance of the client U and the j^{th} instance of the servers S_1 and S_2 are partners and they have all terminated and accepted, then the i^{th} instance of the client and the j^{th} instance of the gateway conclude with the same session key.

A session key is said *fresh* if none of the participants is corrupted, the adversary never asked a Reveal query for the particular instance of either the client or the gateway. We say that the adversary succeeds (event Succ) if he asks a single query $\mathsf{Test}(\mathsf{U}, i, \mathsf{G}, j)$ or $\mathsf{Test}(\mathsf{G}, j, \mathsf{U}, i)$ on a fresh key $\mathsf{sk}^i_{\mathsf{U},\mathsf{G}}$ or $\mathsf{sk}^j_{\mathsf{G},\mathsf{U}}$ and outputs a single bit b' such as b' is equal to the bit b chosen by the Test oracle. His advantage against the protocol π is then defined as $\mathsf{Adv}_{\mathcal{A}}(\pi) = 2\Pr(\mathsf{Succ}) - 1$, the probability being taken over all the random coins used by the adversary and the participants during the execution of the protocol.

The number of on-line attacks are counted by the number of Send queries corresponding to this execution of the protocol. (The $\mathsf{Execute}$ queries are not counted in the on-line attacks.)

Definition 1. *A protocol π is said to be a secure two-server password-authenticated key-exchange if there exists a constant c such that, for all dictionary sizes N and all probabilistic polynomial-time adversaries making at most q_s on-line attacks and corrupting at most one server partnered with a client, $\mathsf{Adv}_{\mathcal{A}}(\pi) \leq \frac{cq_s}{N} + \varepsilon(k)$, where k is the security parameter and ε a negligible function.*

Here, $c = 1$ in our security proofs, which means that the adversary can try one password by execution, *i.e.* on-line dictionary attack (the authors of [KMTG12] only achieve $c = 2$ with their protocol).

B Sketch of Proofs

B.1 Simple Protocol

The proof follows the spirit of the proof given in [KV11, BBC+13]. Recall that at most one server is corrupted during the execution of the protocol (meaning that the adversary cannot trivially recover the password of the client) and all the communication goes through the gateway so that the client apparently interacts with a unique server (it receives a unique message from the gateway (concatenation of the messages sent by the servers) even if it sends messages destined to a set of two servers). One can thus see that the security of our distributed password-authenticated key-exchange, constructed as some sort of three-party PAKE, can be reduced to that of the PAKE proven in the BPR model in [BBC+13]. More precisely, its security can be proven through the following games:

- $\mathbf{Game_0}$: this is the real game, where the adversary has a certain advantage (that we are willing to upper-bound) to win.
- $\mathbf{Game_1}$: we modify the way $\mathsf{Execute}$-queries are answered, by replacing the ciphertexts $\mathcal{C}_i$ by correct encryptions of a random password $\widetilde{\mathsf{pw}}$, so that $\mathcal{C}_i$ is not in $\mathcal{L}_{j,i}$ anymore, for $j \neq i$. This is indistinguishable under the IND-CPA property of the Cramer-Shoup encryption scheme.
- $\mathbf{Game_2}$: we modify the way $\mathsf{Execute}$-queries are answered, by replacing the common session key by a random value. Since the languages are not satisfied from the former game, the smoothness ensures the indistinguishability from the former game.
- $\mathbf{Game_3}$: we modify the way Send_1-queries (when the adversary sent a flow and waits for the answer) are answered, by using a decryption oracle or alternatively knowing the decryption key. If the ciphertext has been generated by the adversary who used the good password, one declares that $\mathcal{A}$ succeeds and terminates the game (event Bad happens). If it is not correct or consistent with the other participant's value, one chooses the session key at random. If it is a replay of a previous flow sent by the simulator, one knows the hashing keys and can compute the session keys using them. The advantage of the adversary only increases if event Bad happens, which probability is computed later on.
- $\mathbf{Game_4}$: we modify again the way Send_1-queries are answered: if the message was already received by a partner of the user, we set the key identical to the key set for this user. Otherwise, the key is set at random. The first case is identical to the previous game since the message is a replay of a previous flow, and the second case is indistinguishable thanks to the technical lemma proven in [KV11].

- **Game$_5$**: we now modify the way Send_0-queries (when the adversary asks a user for a first flow) are answered: as in Game$_1$, one encrypts a random value pw. Since decryptions are now required (to answer Send_1-queries from Game$_3$), this game is indistinguishable under the $\mathsf{IND\text{-}CCA}$ property of the Cramer-Shoup encryption scheme.
- **Game$_6$**: since the hashing and projection keys only depend on public values given in the common reference string, the private values of the honest players are not used anymore in this final game, except for checking whether the adversary has won (event Bad). They can thus be chosen at random, at the very end of the execution, only to check if this event happened. The advantage of the adversary in this last game is thus exactly the probability of this event, which is bounded by q_s/N, with q_s being the number of Send-queries (*i.e.* on-line dictionary attacks) and N the size of the dictionary.

B.2 Efficient Protocol

The proof is conducted exactly as in the former protocol, even simpler since the servers do not need to compute any ciphertext. We thus construct exactly the same security games, except that, when receiving an $\mathsf{Execute}$ or a $\mathsf{Send}(\mathsf{G}, j, m)$ query from the adversary, we only have to compute the hashing keys and send the projected keys on behalf of the servers, since they do not need to compute any ciphertext ($\mathcal{C}_{\mathsf{U}}^{\mathrm{DB}}$ is public). The simulation of the client remains the same. Overall, the advantage of the adversary is bounded by q_s/N, with q_s being the number of Send-queries (*i.e.* on-line dictionary attacks) and N the size of the dictionary.