Smooth Projective Hashing for Conditionally Extractable Commitments

Michel Abdalla, Céline Chevalier, David Pointcheval

This is the full version of the extended abstract that appeared in
Advances in Cryptology – Proceedings of CRYPTO 2009 (16–20 august 2008, Santa Barbara, California, USA)
S. Halevi Ed. Springer-Verlag, LNCS 5677, pages 671–689.

Smooth Projective Hashing for Conditionally Extractable Commitments

Michel Abdalla, Céline Chevalier, and David Pointcheval

École Normale Supérieure, CNRS-INRIA, Paris, France

Abstract. The notion of smooth projective hash functions was proposed by Cramer and Shoup and can be seen as special type of zero-knowledge proof system for a language. Though originally used as a means to build efficient chosen-ciphertext secure public-key encryption schemes, some variations of the Cramer-Shoup smooth projective hash functions also found applications in several other contexts, such as password-based authenticated key exchange and oblivious transfer. In this paper, we first address the problem of building smooth projective hash functions for more complex languages. More precisely, we show how to build such functions for languages that can be described in terms of disjunctions and conjunctions of simpler languages for which smooth projective hash functions are known to exist. Next, we illustrate how the use of smooth projective hash functions with more complex languages can be efficiently associated to extractable commitment schemes and avoid the need for zero-knowledge proofs. Finally, we explain how to apply these results to provide more efficient solutions to two well-known cryptographic problems: a public-key certification which guarantees the knowledge of the private key by the user without random oracles or zero-knowledge proofs and adaptive security for password-based authenticated key exchange protocols in the universal composability framework with erasures.

1 Introduction

In [16], Cramer and Shoup introduced a new primitive called smooth projective hashing and showed how to use it to generalize their chosen-ciphertext secure public-key encryption scheme [15]. The new abstraction not only provided a more intuitive description of the original encryption scheme, but also resulted in several new instantiations based on different security assumptions such as quadratic residuosity and N-residuosity [31].

The notion of smooth projective hash functions (SPHF, [16], after slight modifications [22]) has been proven quite useful and has found applications in several other contexts, such as password-based authenticated key exchange (PAKE, [22]) and oblivious transfer [27]. In the context of PAKE protocols, the work of Gennaro and Lindell abstracted and generalized (under various indistinguishability assumptions) the earlier protocol by Katz, Ostrovsky, and Yung [28] and has become the basis of several other schemes [3, 1, 8]. In the context of oblivious transfer, the work of Kalai [27] also generalized earlier protocols by Naor and Pinkas [30] and by Aiello, Ishai, and Reingold [2].

To better understand the power of SPHF, let us briefly recall what they are. First, the definition of SPHF requires the existence of a domain X and an underlying NP language L such that it is computationally hard to distinguish a random element in L from a random element in $X \setminus L$. For instance, in the particular case of the PAKE scheme in [13], the language L is defined as the set of triples $\{(c, \ell, m)\}$ such that c is an encryption of m with label ℓ under a public key given in the common reference string (CRS). The semantic security of the encryption scheme guarantees computational indistinguishability between elements from L and elements from X.

One of the key properties that make SPHF so useful is that, for a point $x \in L$, the hash value can be computed using either a *secret* hashing key hk, or a *public* projected key hp (depending on x [22] or not [16]) together with a witness w to the fact that $x \in L$. Another important property of these functions is that, given the projected key hp, their output is uniquely defined for points $x \in L$ and statistically indistinguishable from random for points $x \in X \setminus L$. Moreover, without the knowledge of the witness w to the fact that $x \in L$, the output of these functions on x is also pseudo-random.

The first main contribution of this paper is to extend the line of work on SPHF, the element-based version proposed by Gennaro and Lindell [22], to take into account more complex NP languages. More precisely, we show how to build SPHF for languages that can be described in terms of disjunctions and conjunctions of simpler languages for which SPHF are known to exist. For instance, let H_m represent a

family of SPHF for the language $\{(c)\}$, where c is the encryption of m under a given public key. Using our tools, one can build a family of SPHF for the language $\{(c)\}$, where c is the encryption of either 0 or 1, by combining H_0 and H_1.

One of the advantages of building SPHF for more complex languages is that it allows us to simplify the design of the primitives to which they are associated. To demonstrate this, we consider in this paper the specific case of extractable commitment schemes. In most protocols in which extractable commitments are used, the capability of extracting the committed message usually depends on the commitment being properly generated. To achieve this goal and enforce the correct generation of the commitment, it is often the case that additional mechanisms, such as zero-knowledge proofs, may have to be used. This is the case, for instance, of several protocols where a specific public-key registration phase is required, such as most of the cryptographic protocols with dynamic groups (multisignatures [9, 29], group signatures [18], etc). Such a framework is sometimes named *registered public-key model*, where a proof of knowledge of the secret key is required before any certification.

To be able to build more efficient extractable commitment schemes and avoid the use of possibly expensive concurrent zero-knowledge proofs, a second main contribution of this paper is to generalize the concept of extractable commitments so that extraction may fail if the commitment is not properly generated. More specifically, we introduce a new notion of L-extractable commitments in which extraction is only guaranteed if the committed value belongs to the language L and may fail otherwise. The main intuition behind this generalization is that, when used together with a SPHF for the language L, the cases in which extraction may fail will not be very important as the output of the SPHF will be statistically indistinguishable from random in such cases.

1.1 Applications

Registered Public-Key Setting. For many cryptographic protocols, for proving the security even when users can dynamically join the system, the simulator described in the security proof often needs to know the private keys of the authorized users, which is called the *registered public-key setting*, in order to avoid rogue-attacks [9]. This should anyway be the correct way to proceed for a certification authority: it certifies a public key to a user if and only if the latter provides a proof of knowledge of the associated private key. However, in order to allow concurrency, intricate zero-knowledge proofs are required, which makes the certification process either secure in the random oracle model [6] only, or inefficient in the standard model.

In this paper, we show how SPHF with conditionally extractable commitments can help to solve this problem efficiently, in the standard model, by establishing a secure channel between the players, with keys that are either the same for the two parties if the commitment has been correctly built, or perfectly independent in the other case.

Adaptively-secure PAKE schemes. We thereafter study more involved key exchange schemes. In 1992, Bellovin and Merritt [7] suggested a method to authenticate a key exchange based on simple passwords, possibly drawn from a space so small that an adversary might enumerate off-line all possible values. Because of the practical interest of such a primitive, many schemes have been proposed and studied. In 2005, Canetti *et al.* [13] proposed an ideal functionality for PAKE protocols, in the universal composability (UC) framework [11, 14], and showed how a simple variant of the Gennaro-Lindell methodology [22] could lead to a secure protocol. Though quite efficient, their protocol is not known to be secure against adaptive adversaries, where they can corrupt players at any time, and learn their internal states. The first ones to propose an adaptively-secure PAKE in the UC framework were Barak *et al.* [3] using general techniques from multi-party computation (MPC). Though conceptually simple, their solution yields quite inefficient schemes.

Here, we take a different approach. Instead of using general MPC techniques, we extend the Gennaro-Lindell methodology to deal with adaptive corruptions by using a non-malleable conditionally-extractable and equivocable commitment scheme with an associated SPHF family. The new scheme is adaptively secure in the common reference string model in the UC framework under standard complexity assumptions with erasures.

1.2 Related work

Commitments. Commitment schemes are one of the most fundamental cryptographic primitives, being used in several cryptographic applications such as zero-knowledge proofs [25] and secure multi-party computation [24]. Even quite practical protocols need them, as already explained above in the public-key registration setting, but also in password-based authenticated key exchange [22]. They allow a user to commit a value x into a public value C, such that the latter does not reveal any information about x (the hiding property), but C can be opened later to x only: one cannot change its mind (the binding property). Various additional properties are often required, such as non-malleability, extractability and equivocability. Canetti and Fischlin [12] provided an ideal functionality for such a primitive and showed that achieving all these properties at the same time was impossible in the UC plain model. They also provided the first candidate in the CRS model. Damgård and Nielsen [17] later proposed another construction of universally composable commitments, that is more efficient for some applications. Since we want to avoid the use of possibly inefficient proofs of relations present in the Damgård-Nielsen construction and given that the Canetti-Fischlin construction is well suited for our purpose of designing an associated smooth hash function, we opted to use the latter as the starting point for our constructions.

PAKE. The password-based setting was first considered by Bellovin and Merritt [7] and followed by many proposals. In 2000, Bellare, Pointcheval, and Rogaway [5] as well as Boyko, MacKenzie, and Patel [10] proposed security models and proved variants of the Bellovin and Merritt protocol [7], under ideal assumptions, such as the random oracle model [6]. Soon after, Katz, Ostrovsky, and Yung [28] and Goldreich and Lindell [23] proposed the first protocols with a proof of security in the standard model, with the former being based on the decisional Diffie-Hellman assumption and the latter on general assumptions. Later, Gennaro and Lindell [22] proposed an abstraction and generalization of the KOY protocol and became the basis of several other variants, including ours in the last section.

1.3 Organization of the Paper

In Section 2, we review the basic primitives needed in this paper. Then, in Section 3, we describe our first contribution: SPHF families on conjunctions and disjunctions of languages. In Section 4 we combine that with our second contribution, conditionally-extractable commitments. We focus on the ElGamal-based commitment, since this is enough to build more efficient public-key certification protocols. Finally, in Section 5, we add equivocability to the commitment, borrowing techniques from Canetti and Fischlin [12]. Then, we add the non-malleability property, granted the Cramer-Shoup encryption scheme, which can then be used to build an adaptively-secure PAKE in the UC framework, based on the Gennaro and Lindell [22] framework. Due to space restrictions, formal definitions, proofs, and application details were postponed to the appendix.

2 Commitments

In the following, we focus on Pedersen commitments, and certification of Schnorr-like public keys, hence, we work in the discrete logarithm setting. As a consequence, to get extractable commitments, we use encryption schemes from the same family: the ElGamal encryption [21] and the labeled version of the Cramer-Shoup encryption scheme [15] (for achieving non-malleability).

Labeled Public-Key Encryption. Labeled encryption [32] is a variation of the usual encryption notion that takes into account the presence of labels in the encryption and decryption algorithms. More precisely, both the encryption and decryption algorithms have an additional input parameter, referred to as a label, and the decryption algorithm should only correctly decrypt a ciphertext if its input label matches the label used to create that ciphertext.

The security notion for labeled encryption is similar to that of standard encryption schemes. The main difference is that, whenever the adversary wishes to ask a query to its Left-or-Right encryption

oracle in the indistinguishability security game (IND-CPA) [26, 4], in addition to providing a pair of messages (m_0, m_1), it also has to provide a target label ℓ to obtain the challenge ciphertext c. When chosen-ciphertext security (IND-CCA) is concerned, the adversary is also allowed to query its decryption oracle on any pair (ℓ', c') as long as $\ell' \neq \ell$ or the ciphertext c' does not match the output c of a query to its Left-or-Right encryption oracle whose input includes the label ℓ. For formal security definitions for labeled encryption schemes, please refer to [13, 1].

One of the advantages of using labeled encryption, which we exploit in this paper, is that we can easily combine several IND-CCA labeled encryption schemes with the help of a strongly unforgeable one-time signature scheme so that the resulting scheme remains IND-CCA [20].

ElGamal and Cramer-Shoup Encryption. We denote by G a cyclic group of prime order q where q is large (n bits), and g a generator for this group. Let $\mathsf{pk} = (g_1, g_2, c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2}, h = g_1^z, H)$ be the public key of the Cramer-Shoup scheme, where g_1 and g_2 are random group elements, x_1, x_2, y_1, y_2 and z are random scalars in $\mathbb{Z}_q$, and H is a collision-resistant hash function (actually, second-preimage resistance is enough), and $\mathsf{sk} = (x_1, x_2, y_1, y_2, z)$ the associated private key. Note that (g_1, h) will also be seen as the public key of the ElGamal encryption, with z the associated private key. For the sake of simplicity, we assume in the following that public keys will additionally contain all the global parameters, such as the group G.

If $M \in G$, the multiplicative ElGamal encryption is defined as $\mathsf{EG}^{\times}_{\mathsf{pk}}(M; r) = (u_1 = g_1^r, e = h^r M)$, which can be decrypted by $M = e/u_1^z$. If $M \in \mathbb{Z}_q$, the additive ElGamal encryption is defined as $\mathsf{EG}^{+}_{\mathsf{pk}}(M; r) = (u_1 = g_1^r, e = h^r g^M)$. Note that $\mathsf{EG}^{\times}_{\mathsf{pk}}(g^M; r) = \mathsf{EG}^{+}_{\mathsf{pk}}(M; r)$. It can be decrypted after an additional discrete logarithm computation: M must be small enough. Similarly, if $M \in G$, the multiplicative labeled Cramer-Shoup encryption is defined as $\mathsf{CS}^{\times\,\ell}_{\mathsf{pk}}(M; r) = (u_1, u_2, e, v)$, such that $u_1 = g_1^r$, $u_2 = g_2^r$, $e = h^r M$, $\theta = H(\ell, u_1, u_2, e)$ and $v = (cd^\theta)^r$. Decryption works as above, with $M = e/u_1^z$, but only if the ciphertext is valid: $v = u_1^{x_1+\theta y_1} u_2^{x_2+\theta y_2}$. If $M \in \mathbb{Z}_q$, its additive encryption $\mathsf{CS}^{+\,\ell}_{\mathsf{pk}}(M; r)$ is such that $e = h^r g^M$. The following relation holds $\mathsf{CS}^{\times\,\ell}_{\mathsf{pk}}(g^M; r) = \mathsf{CS}^{+\,\ell}_{\mathsf{pk}}(M; r)$. The decryption applies as above if M is small enough.

As already noted, from any Cramer-Shoup ciphertext (u_1, u_2, e, v) of a message M with randomness r, whatever the label ℓ is, one can extract (u_1, e) as an ElGamal ciphertext of the same message M with the same randomness r. This extraction applies independently of the additive or multiplicative version since the decryption works the same for the ElGamal and the Cramer-Shoup ciphertexts, except for the validity check that provides the CCA security level to the Cramer-Shoup encryption scheme, whereas the ElGamal encryption scheme achieves IND-CPA security level only.

Commitments. With a commitment scheme, a player can commit to a secret value x by publishing a commitment $C = \mathsf{com}(x; r)$ with randomness r, in such a way that C reveals nothing about the secret x, which is called the *hiding* property. The player can later open C to reveal x, by publishing x and a decommitment, also referred to as witness, in a publicly verifiable way: the player cannot open C to any other value than x, which is the *binding* property. In many cases, the decommitment consists of the random r itself or some part of it. In this paper, we only consider commitment schemes in the common reference string (CRS) model in which the common parameters, referred to as the CRS, are generated honestly and available to all parties.

Note that an IND-CPA public-key encryption scheme provides such a commitment scheme: the binding property is guaranteed by the uniqueness of the plaintext (perfectly binding), and the hiding property is guaranteed by the IND-CPA security (computationally hiding). In this case, the CRS simply consists of the public-key of the encryption scheme. The Pedersen commitment $C = \mathsf{comPed}(x; r) = g^x h^r$ provides a perfectly hiding, but computationally binding commitment under the intractability of the discrete logarithm of h in basis g.

We now present additional properties that can be satisfied by the commitment. First, we say that a commitment is *extractable* if there exists an efficient algorithm, called an extractor, capable of generating a new set of common parameters (*i.e.*, a new CRS) whose distribution is equivalent to that of an honestly generated CRS and such that it can extract the committed value x from any commitment

C. This is of course only possible for computationally hiding commitments, such as encryption schemes: the decryption key is the extraction trapdoor. Second, we say that a commitment is *equivocable* if there exists an efficient algorithm, called an equivocator, capable of generating a new CRS and a commitment with similar distributions to those of the actual scheme and such that the commitment can be opened in different ways. Again, this is possible for computationally binding commitments only, such as the Pedersen commitment: the knowledge of the discrete logarithm of h in basis g is a trapdoor that allows the opening of a commitment in more than one way. Finally, a *non-malleable* commitment ensures that if an adversary that receives a commitment C of some unknown value x can generate a valid commitment for a related value y, then a simulator could perform as well without seeing the commitment C. A public-key encryption scheme that is IND-CCA provides such a non-malleable commitment [22]. For formal security definitions for commitment schemes, please refer to [22, 19, 12].

In the following, we use encryption schemes in order to construct commitments, which immediately implies the hiding, binding and extractable properties, as said above. However, when one uses the additive versions of ElGamal or Cramer-Shoup encryption schemes, extractability (or decryption) is only possible if the committed values (or plaintexts) are small enough, hence our notion of L-extractable commitments (see Section 4) which will mean that the commitment is extractable if the committed value lies in the language L. More precisely, we will split the value to be committed in small pieces (that lie in the language L), but we will then need to be sure that they actually lie in this language to guarantee extractability. We thus introduce smooth hash functions in order to allow communications if the commitments are valid only.

3 Smooth Hash Functions on Conjunctions and Disjunctions of Languages

3.1 Smooth Projective Hash Functions.

Projective hash function families were first introduced by Cramer and Shoup [16] as a means to design chosen-ciphertext secure encryption schemes. We here use the definitions of Gennaro and Lindell [22], who later showed how to use such families to build secure password-based authenticated key exchange protocols, together with non-malleable commitments. In addition to commitment schemes, we also consider here families of SPHF associated to labeled encryption as done by Canetti *et al.* [13] and by Abdalla and Pointcheval [1].

Let X be the domain of these functions and let L be a certain subset of points of this domain (a language). A key property of these functions is that, for points in L, their values can be computed by using either a *secret* hashing key or a *public* projected key. While the computation using the *secret* hashing key works for all points in the domain X of the hash function, the computation using a *public* projected key only works for points $x \in L$ and requires the knowledge of the witness w to the fact that $x \in L$. A projective hash function family is said to be *smooth* if the value of the function on inputs that are outside the particular subset L of the domain are independent of the projected key. Another important property of these functions is that, given the projected key hp, their output is uniquely defined for points $x \in L$. Moreover, if L is a *hard partitioned subset* of X (*i.e.*, it is computationally hard to distinguish a random element in L from a random element in $X \setminus L$), this output is also *pseudo-random* if one does not know a witness w to the fact that $x \in L$ [22]. The interested reader is referred to Appendix A for more formal definitions.

In the particular case of the Gennaro-Lindell scheme [22], the subset $L_{\mathsf{pk},m}$ was defined as the set of $\{(c)\}$ such that c is a commitment of m using public parameters pk: there exists r for which $c = \mathsf{com}_{\mathsf{pk}}(m; r)$ where com is the committing algorithm of the commitment scheme. In the case of the CHKLM scheme [13], the subset $L_{\mathsf{pk},(\ell,m)}$ was defined as the set of $\{(c)\}$ such that c is an encryption of m with label ℓ, under the public key pk: there exists r for which $c = \mathcal{E}^{\ell}_{pk}(m; r)$ where $\mathcal{E}$ is the encryption algorithm of the labeled encryption scheme. In the case of a standard encryption scheme, the label is simply omitted. The interested reader is referred to [22, 13, 1] for more details.

Languages. Since we want to use more general languages, we need more detailed notations. Let **LPKE** be a labeled encryption scheme with public key pk. Let X be the range of the encryption algorithm. Here are three useful examples of languages L in X:

- the valid ciphertexts c of m under pk, $L_{(\mathbf{LPKE},\mathsf{pk}),(\ell,m)} = \{c | \exists r\ c = \mathcal{E}^{\ell}_{\mathsf{pk}}(m;r)\}$;
- the valid ciphertexts c of m_1 or m_2 under pk (that is, a disjunction of two versions of the former languages), $L_{(\mathbf{LPKE},\mathsf{pk}),(\ell,m_1 \vee m_2)} = L_{(\mathbf{LPKE},\mathsf{pk}),(\ell,m_1)} \cup L_{(\mathbf{LPKE},\mathsf{pk}),(\ell,m_2)}$;
- the valid ciphertexts c under pk, $L_{(\mathbf{LPKE},\mathsf{pk}),(\ell,*)} = \{c | \exists m\ \exists r\ c = \mathcal{E}^{\ell}_{\mathsf{pk}}(m;r)\}$.

If the encryption scheme is IND-CPA, the first two are hard partitioned subsets of X. The last one can also be a hard partitioned subset in some cases: for the Cramer-Shoup encryption, $L \subsetneq X = G^4$ and, in order to distinguish a valid ciphertext from an invalid one, one has to break the DDH problem. However, for the ElGamal encryption scheme, all the ciphertexts are valid, hence $L = X = G^2$.

More complex languages can be defined, with disjunctions as above, or conjunctions: the pairs of ciphertexts (a, b) such that $a \in L_{(\mathbf{LPKE},\mathsf{pk}),(\ell,0\vee1)}$ and $b \in L_{(\mathbf{LPKE},\mathsf{pk}),(\ell,2\vee3)}$. This set can be obtained by $(L_{(\mathbf{LPKE},\mathsf{pk}),(\ell,0\vee1))} \times X) \cap (X \times L_{(\mathbf{LPKE},\mathsf{pk}),(\ell,2\vee3)})$.

Likewise, we can define more general languages based on other primitives such as commitment schemes. The definition would be similar to the one above, with pk playing the role of the common parameters, $\mathcal{E}_{\mathsf{pk}}$ playing the role of the committing algorithm, (m, ℓ) playing the role of the input message, and c playing the role of the commitment.

More generally, in the following, we denote the language by the generic notation $L_{(\mathbf{Sch},\rho),aux}$ where aux denotes all the parameters useful to characterize the language (such as the label used, or a plaintext), ρ denotes the public parameters such as a public key pk, and **Sch** denotes the primitive used to define the language, such as an encryption scheme **LPKE** or a commitment scheme **Com**. When there is no ambiguity, the associated primitive **Sch** will be omitted.

We now present new constructions of SPHF to deal with more complex languages, such as disjunctions and conjunctions of any languages. The constructions are presented for two languages but can be easily extended to any polynomial number of languages. We then discuss about possible information leakage at the end of this section. The properties of correctness, smoothness and pseudo-randomness are easily verified by these new smooth hash systems. Due to the lack of space, the formal proofs can be found in Appendix B.

3.2 Conjunction of two Generic Smooth Hashes

Let us consider an encryption or commitment scheme defined by public parameters and a public key aggregated in ρ. X is the range of the elements we want to study (ciphertexts, tuples of ciphertexts, commitments, etc), and $L_1 = L_{1,\rho,aux}$ and $L_2 = L_{2,\rho,aux}$ are hard partitioned subsets of X, which specify the expected properties (valid ciphertexts, ciphertexts of a specific plaintext, etc). We consider situations where X possesses a group structure, which is the case if we consider ciphertexts or tuples of ciphertexts from an homomorphic encryption scheme. We thus denote by $\oplus$ the commutative law of the group (and by $\ominus$ the opposite operation, such that $c \oplus a \ominus a = c$).

We assume to be given two smooth hash systems SHS_1 and SHS_2, on the sets corresponding to the languages L_1 and L_2: $\mathsf{SHS}_i = \{\mathsf{HashKG}_i, \mathsf{ProjKG}_i, \mathsf{Hash}_i, \mathsf{ProjHash}_i\}$. Here, HashKG_i and ProjKG_i denote the hashing key and the projected key generators, and Hash_i and $\mathsf{ProjHash}_i$ the algorithms that compute the hash function using hk_i and hp_i respectively.

Let c be an element of X, and r_1 and r_2 two elements chosen at random. We denote the keys by

$$\mathsf{hk}_1 = \mathsf{HashKG}_1(\rho, aux, r_1), \quad \mathsf{hk}_2 = \mathsf{HashKG}_2(\rho, aux, r_2)$$

and

$$\mathsf{hp}_1 = \mathsf{ProjKG}_1(\mathsf{hk}_1; \rho, aux, c), \quad \mathsf{hp}_2 = \mathsf{ProjKG}_2(\mathsf{hk}_2; \rho, aux, c).$$

A smooth hash system for the language $L = L_1 \cap L_2$ is then defined as follows, if $c \in L_1 \cap L_2$ and w_i is a witness that $c \in L_i$, for $i = 1, 2$:

$$\mathsf{HashKG}_L(\rho, aux, r = r_1 \| r_2) = \mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2) \qquad \mathsf{ProjKG}_L(\mathsf{hk}; \rho, aux, c) = \mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2)$$
$$\mathsf{Hash}_L(\mathsf{hk}; \rho, aux, c) = \mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, c) \oplus \mathsf{Hash}_2(\mathsf{hk}_2; \rho, aux, c)$$
$$\mathsf{ProjHash}_L(\mathsf{hp}; \rho, aux, c; (w_1, w_2)) = \mathsf{ProjHash}_1(\mathsf{hp}_1; \rho, aux, c; w_1) \oplus \mathsf{ProjHash}_2(\mathsf{hp}_2; \rho, aux, c; w_2)$$

3.3 Disjunction of two Generic Smooth Hashes

Let L_1 and L_2 be two languages as described above. We assume to be given two smooth hash systems SHS_1 and SHS_2 with respect to these languages. We define $L = L_1 \cup L_2$ and construct a smooth projective hash function for this language as follows:

$$\begin{aligned}
\mathsf{HashKG}_L(\rho, aux, r = r_1 \| r_2) &= \mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2) \\
\mathsf{ProjKG}_L(\mathsf{hk}; \rho, aux, c) &= \mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2, \mathsf{hp}_\Delta = \mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, c) \oplus \mathsf{Hash}_2(\mathsf{hk}_2; \rho, aux, c)) \\
\mathsf{Hash}_L(\mathsf{hk}; \rho, aux, c) &= \mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, c) \\
\mathsf{ProjHash}_L(\mathsf{hp}; \rho, aux, c; w) &= \mathsf{ProjHash}_1(\mathsf{hp}_1; \rho, aux, c; w) \quad \text{if } c \in L_1 \\
&\text{or } \mathsf{hp}_\Delta \ominus \mathsf{ProjHash}_2(\mathsf{hp}_2; \rho, aux, c; w) \quad \text{if } c \in L_2
\end{aligned}$$

where w is a witness of $c \in L_i$ for $i \in \{1, 2\}$. Then $\mathsf{ProjHash}_i(\mathsf{hp}_i; \rho, aux, c; w) = \mathsf{Hash}_i(\mathsf{hk}_i; \rho, aux, c)$. The player in charge of computing this value is supposed to know the witness, and in particular the language which c belongs to (and thus the index i).

3.4 Uniformity and Independence

In the above definition of SPHF (contrarily to the original Cramer-Shoup [16] definition), the value of the projected key formally depends on the ciphertext/commitment c. However, in some cases, one may not want to reveal any information about this dependency. In fact, in certain cases such as in the construction of a SPHF for equivocable and extractable commitments in Section 5, one may not even want to leak any information about the auxiliary elements aux. When no information is revealed about aux, it means that the details about the exact language will be concealed.

We thus add a notion similar to the smoothness, but for the projected key: the projected key may or may not depend on c (and aux), but its distribution does not: Let us denote by $D_{\rho,aux,c}$ the distribution $\{\mathsf{hp} \mid \mathsf{hk} = \mathsf{HashKG}_L(\rho, aux, r) \text{ and } \mathsf{hp} = \mathsf{ProjKG}_L(\mathsf{hk}; \rho, aux, c)\}$, on the projected keys. If, for any $c, c' \in X$, $D_{\rho,aux,c'}$ and $D_{\rho,aux,c}$ are indistinguishable, then we say that the smooth hash system has the *1-uniformity* property. If, for any $c, c' \in X$, and any auxiliary elements aux, aux', $D_{\rho,aux',c'}$ and $D_{\rho,aux,c}$ are indistinguishable, we name it *2-uniformity* property.

More than indistinguishability of distributions, the actual projected key hp may not depend at all on c, as in the Cramer and Shoup's definition. Then, we say that the smooth hash system guarantees *1-independence* (resp. *2-independence* if it does not depend on aux either). Note that the latter independence notions immediately imply the respective uniformity notions.

As an example, the smooth hash system associated with the ElGamal cryptosystem (see Section 4.1) guarantees 2-independence. On the other hand, the analogous system associated with the Cramer-Shoup encryption (see Appendix D.1) guarantees 2-uniformity only. For smooth hash systems combinations, one can note that in the case of disjunctions, one can get, at best, the uniformity property, since hash computations on the commitment are needed for generating the projected key. Furthermore, this is satisfied under the condition that the two underlying smooth hash systems already satisfy this property (see Appendix B for more details and proofs).

Finally, one should note that, in the case of disjunction, the view of the projected hash value could leak some information about the sub-language in which the input lies, if an adversary sends a fake hp_Δ. The adversary could indeed check whether $\mathsf{ProjHash}_L(\mathsf{hp}; \rho, aux, c; w)$ equals $\mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, c)$ or $\mathsf{hp}_\Delta \ominus \mathsf{Hash}_2(\mathsf{hk}_2; \rho, aux, c)$. But first, it does not contradict any security notion for smooth hash systems; second, in all the applications below, the projected hash value is never revealed; and third, in

the extractable commitments below, because of the global conjunction of the languages, an exponential exhaustive search would be needed to exploit this information, even if the committed value is a low-entropy one.

4 A Conditionally Extractable Commitment

4.1 ElGamal Commitment and Associated Smooth Hash

The ElGamal commitment is realized in the common reference string model, where the CRS ρ contains (G, pk), as defined in Section 2, for the ElGamal encryption scheme. In practice, sk should not be known by anybody, but in the security analysis, sk will be the extraction trapdoor. Let the input of the committing algorithm be a scalar $M \in \mathbb{Z}_q$. The commitment algorithm consists of choosing a random r and computing the following ElGamal encryption under random r: $C = \mathsf{EG}^+_{\mathsf{pk}}(M, r) = (u_1 = g_1^r, e = h^r g^M)$.

The smooth projective hashing, associated with this commitment scheme and the language $L = L_{(\mathbf{EG}^+,\rho),M} \subset X = G^2$ of the additive ElGamal ciphertexts C of M under the global parameters and public key defined by ρ, is the family based on the underlying ElGamal encryption scheme, as defined in [22]:

$$\mathsf{HashKG}((\mathbf{EG}^+,\rho),M) = \mathsf{hk} = (\gamma_1,\gamma_3) \xleftarrow{\$} \mathbb{Z}_q \times \mathbb{Z}_q \qquad \mathsf{Hash}(\mathsf{hk};(\mathbf{EG}^+,\rho),M,C) = (u_1)^{\gamma_1}(eg^{-M})^{\gamma_3}$$
$$\mathsf{ProjKG}(\mathsf{hk};(\mathbf{EG}^+,\rho),M,C) = \mathsf{hp} = (g_1)^{\gamma_1}(h)^{\gamma_3} \qquad \mathsf{ProjHash}(\mathsf{hp};(\mathbf{EG}^+,\rho),M,C;r) = (\mathsf{hp})^r$$

First, under the Decisional Diffie-Hellman problem (semantic security of the ElGamal encryption scheme), L is a hard partitioned subset of $X = G^2$. Then, for $C = \mathsf{EG}^+_{\mathsf{pk}}(M, r)$, and thus with the witness r, the algorithms are defined as above using the same notations as in [22].

4.2 L-extractable Commitments

Note that the value g^M would be easily extractable from this commitment (seen as the multiplicative ElGamal encryption). However, one can extract M itself (the actual committed value) only if its size is small enough so that it can be found as a solution to the discrete logarithm problem. In order to obtain "extractability" (up to a certain point, see below), one should rather commit to it in a bit-by-bit way.

Let us denote $M \in \mathbb{Z}_q$ by $\sum_{i=1}^m M_i \cdot 2^{i-1}$, where $m \leq n$. Its commitment is $\mathsf{comEG}_{\mathsf{pk}}(M) = (b_1, \ldots, b_m)$, where $b_i = \mathsf{EG}^+_{\mathsf{pk}}(M_i \cdot 2^{i-1}, r_i) = (u_{1,i} = g_1{}^{r_i}, e_i = h^{r_i} g^{M_i \cdot 2^{i-1}})$, for $i = 1, \ldots, m$. The homomorphic property of the encryption scheme allows to obtain, from this tuple, the above simple commitment of M

$$C = \mathsf{EG}^+_{\mathsf{pk}}(M, r) = (u_1, e) = (\textstyle\prod u_{1,i}, \prod e_i) = \prod b_i, \text{ for } r = \sum r_i.$$

We now precise what we mean by "extractability": Here, the commitment will be extractable if the messages M_i are bits (or at least small enough), but we cannot ensure that it will be extractable otherwise. More generally, this leads to a new notion of *$L-$extractable commitments*, which means that we allow the primitive not to be extractable if the message does not belong to a certain language L (*e.g.* the language of encryptions of 0 or 1), which is informally the language of all commitments valid and "of good shape", and is included into the set X of all commitments.

Smooth Hash Functions. For the above protocol, we need a smooth hash system on the language $L = L_1 \cap L_2$, where $L_1 = \{(b_1, \ldots, b_m) \mid \forall i, b_i \in L_{(\mathbf{EG}^+,\rho),0\vee 1}\}$, $L_2 = \{(b_1, \ldots, b_m) \mid C = \prod_i b_i \in L_{(\mathbf{EG}^\times,\rho),g^M}\}$, to within a factor (corresponding to the offest 2^{i-1}) with

$$L_{(\mathbf{EG}^+,\rho),0\vee 1} = L_{(\mathbf{EG}^+,\rho),0} \cup L_{(\mathbf{EG}^+,\rho),1} \qquad L_{(\mathbf{EG}^+,\rho),0} = \{C \mid \exists r\ C = \mathsf{EG}^+_{\mathsf{pk}}(0,r)\}$$
$$L_{(\mathbf{EG}^\times,\rho),g^M} = \{C \mid \exists r\ C = \mathsf{EG}^\times_{\mathsf{pk}}(g^M,r)\} \quad L_{(\mathbf{EG}^+,\rho),1} = \{C \mid \exists r\ C = \mathsf{EG}^+_{\mathsf{pk}}(1,r)\}$$

It is easy to see that this boils down to constructing a smooth hash system corresponding to a conjunction and disjunction of languages, as presented in the previous section.

4.3 Certification of Public Keys

Description. A classical application of extractable commitments is in the certification of public keys (when we want to be sure that a person joining the system actually knows the associated private key). Suppose that a user U owns a pair of secret and public keys, and would like to have the public key certified by the authority. A natural property is that the authority will not certify this public key unless it is sure that the user really owns the related private key, which is usually ensured by a zero-knowledge proof of knowledge: the user knows the private key if a successful extractor exists.

Here we present a construction that possesses the same property without requiring any explicit proof of knowledge, furthermore in a concurrent way since there is no need of any rewinding:

- First, the user sends his public key g^M, along with a bit-by-bit L-extractable commitment of the private key M, i.e. a tuple $\mathsf{comEG}_{\mathsf{pk}}(M) = (b_1, \ldots, b_m)$ as described above, from which one can derive $C = \prod b_i = \mathsf{EG}^{+}_{\mathsf{pk}}(M, r) = \mathsf{EG}^{\times}_{\mathsf{pk}}(g^M, r)$.
- We define the smooth hash system related to the language $L_1 \cap L_2$, where $L_1 = \cap_i L_{1,i}$, with $L_{1,i}$ the language of the tuples where the i-th component b_i is an encryption of 0 or 1, and L_2 is the language of the tuples where the derived $C = \prod b_i$ is an encryption of the public key g^M (under the multiplicative ElGamal, as in Section 4.1).
  Note that when the tuple $(b_1, \ldots, b_m)$ lies in $L_1 \cap L_2$, it really corresponds to an extractable commitment of the private key M associated to the public key g^M: each b_i encrypts a bit, and can thus be decrypted, which provides the i-th bit of M.
- The authority computes a hash key hk, the corresponding projected key hp on $(b_1, \ldots, b_m)$ and the related hash value Hash on $(b_1, \ldots, b_m)$. It sends hp to U along with $\mathsf{Cert} \oplus \mathsf{Hash}$, where Cert is the expected certificate. Note that if Hash is not large enough, a pseudo-random generator can be used to expand it.
- The user is then able to recover his certificate if and only if he can compute Hash: this value can be computed with the algorithm $\mathsf{ProjHash}$ on $(b_1, \ldots, b_m)$, from hp. But it also requires a witness w proving that the tuple $(b_1, \ldots, b_m)$ lies in $L_1 \cap L_2$.

With the properties of the smooth hash system, if the user correctly computed the commitment, he knows the witness w, and can get the same mask Hash to extract the certificate. If the user cheated, the smoothness property makes Hash perfectly unpredictable: no information is leaked about the certificate.

Security Analysis. Let us outline the security proof of the above protocol. First, the security model is the following: no one can obtain a certificate on a public key if it does not know the associated private key (that is, if no simulator can extract the private key). In other words, the adversary wins if it is able to output (g^M, Cert) and no simulator can produce M.

The formal attack game can thus be described as follows: the adversary $\mathcal{A}$ interacts several times with the authority, by sending public keys and commitments, and asks for the corresponding certificates. It then outputs a pair (g^M, Cert) and wins if no simulator is able to extract M from the transcript.

The simulator works as follows: it is given access to a certification (signing) oracle, and generates a pair of public and private keys $(\mathsf{sk}, \mathsf{pk})$ for the ElGamal encryption. The public key is set as the CRS that defines the commitment scheme. The private key will thus be the extraction trapdoor.

When the simulator receives a certification request, with a public key and a commitment, it first tries to extract the associated private key, granted the extraction trapdoor. In case of success, the simulator asks the signing oracle to provide it with the corresponding certificate on the public key, and complete the process as described in the protocol. However, extraction may fail if the commitments are not well constructed (not in $L_1 \cap L_2$). In such a case, the simulator sends back a random bit-string of appropriate length. In case of successful extraction, the answer received by the user is exactly the expected one. In case of failure, it is perfectly indistinguishable too since the smoothness property of the hash function would make a perfectly random mask Hash (since the input is not in the language).

After several interactions, $\mathcal{A}$ outputs a pair (g^M, Cert), which is forwarded by the simulator. Either g^M has been queried to the signing oracle, which means that the extraction had succeeded, the simulator

knows M and the adversary did not win the attack game, or this is a valid signature on a new message: existential forgery under chosen-message attack.

5 A Conditionally Extractable Equivocable Commitment

In this section, we enhance the previous commitment schemes with equivocability, which is not a trivial task when one wants to keep the extraction property. Note that we first build a malleable extractable and equivocable commitment using the ElGamal-based commitment (see Section 4.1), but one can address the non-malleability property by simply building the commitment upon the Cramer-Shoup encryption scheme. All the details of this extension are given in Appendix D. In the following, if b is a bit, we denote its complement by $\overline{b}$ (i.e., $\overline{b} = 1 - b$). We furthermore denote by $x[i]$ the i^{th} bit of the bit-string x.

5.1 Equivocability

Commitments that are both extractable and equivocable seem to be very difficult to obtain. Canetti and Fischlin [12] proposed a solution but for one bit only. Damgård and Nielsen [17] proposed later another construction. But for efficiency reasons, in our specific context, we extend the former proposal. In this section, we thus enhance our previous commitment (that is already L-extractable) to make it equivocable, using the Canetti and Fischlin's approach. Section 5.3 will then apply a non-malleable variant of our new commitment together with the associated smooth hash function family in order to build a password-authenticated key exchange protocol with adaptive security in the UC framework [11]. The resulting protocol is reasonably efficient and, in particular, more efficient than the protocol by Barak *et al.* [3], which to our knowledge is the only one achieving the same level of security in the standard model.

Description of the Commitment. Our commitment scheme is a natural extension of Canetti-Fischlin commitment scheme [12], in a bit-by-bit way. It indeed uses the ElGamal public-key encryption scheme, for each bit of the bit-string. Let $(y_1, \ldots, y_m)$ be random elements in G. This commitment is realized in the common reference string model, the CRS ρ contains (G, pk), where pk is an ElGamal public key and the private key is unknown to anybody, except to the commitment extractor. It also includes this tuple $(y_1, \ldots, y_m)$, for which the discrete logarithms in basis g are unknown to anybody, except to the commitment equivocator. Let the input of the committing algorithm be a bit-string $\pi = \sum_{i=1}^{m} \pi_i \cdot 2^{i-1}$. The algorithm works as follows:

- For $i = 1, \ldots, m$, it chooses a random value $x_{i,\pi_i} = \sum_{j=1}^{n} x_{i,\pi_i}[j] \cdot 2^{j-1}$ and sets $x_{i,\overline{\pi}_i} = 0$.
- For $i = 1, \ldots, m$, the algorithm commits to π_i, using the random x_{i,π_i}: $a_i = \mathsf{comPed}(\pi_i, x_{i,\pi_i}) = g^{x_{i,\pi_i}} y_i^{\pi_i}$ and defining $\mathbf{a} = (a_1, \ldots, a_m)$.
- For $i = 1, \ldots, m$, it computes the ElGamal commitments (see the previous section) of $x_{i,\delta}$, for $\delta = 0, 1$: $(\mathbf{b}_{i,\delta} = (b_{i,\delta}[j])_j = \mathsf{comEG}_{\mathsf{pk}}(x_{i,\delta})$, where $b_{i,\delta}[j] = \mathsf{EG}^{+}_{\mathsf{pk}}(x_{i,\delta}[j] \cdot 2^{j-1}, r_{i,\delta}[j])$. One can directly extract from the computation of the $b_{i,\delta}[j]$ an encryption $B_{i,\delta}$ of $x_{i,\delta}$: $B_{i,\delta} = \prod_j b_{i,\delta}[j] = \mathsf{EG}^{+}_{\mathsf{pk}}(x_{i,\delta}, r_{i,\delta})$, where $r_{i,\delta}$ is the sum of the random coins $r_{i,\delta}[j]$.

The entire random string for this commitment is (where n is the bit-length of the prime order q of the group G) $R = (x_{1,\pi_1}, (r_{1,0}[1], r_{1,1}[1], \ldots, r_{1,0}[n], r_{1,1}[n]), \ldots, x_{m,\pi_m}, (r_{m,0}[1], \ldots, r_{m,1}[n]))$. From which, all the values $r_{i,\overline{\pi}_i}[j]$ can be erased, letting the opening data (witness of the committed value) become limited to $\mathsf{w} = (x_{1,\pi_1}, (r_{1,\pi_1}[1], \ldots, r_{1,\pi_1}[n]), \ldots, x_{m,\pi_m}, (r_{m,\pi_m}[1], \ldots, r_{m,\pi_m}[n]))$. The output of the committing algorithm, of the bit-string π, using the random R, is $\mathsf{com}_\rho(\pi; R) = (\mathbf{a}, \mathbf{b})$, where $\mathbf{a} = (a_i = \mathsf{comPed}(\pi_i, x_{i,\pi_i}))_i$, $\mathbf{b} = (b_{i,\delta}[j] = \mathsf{EG}^{+}_{\mathsf{pk}}(x_{i,\delta}[j] \cdot 2^{j-1}, r_{i,\delta}[j]))_{i,\delta,j}$.

Opening. In order to open this commitment to π, the above witness w (with the value π) is indeed enough: one can build again, for all i and j, $b_{i,\pi_i}[j] = \mathsf{EG}^{+}_{\mathsf{pk}}(x_{i,\pi_i}[j] \cdot 2^{j-1}, r_{i,\pi_i}[j])$, and check them with $\mathbf{b}$. One can then also compute again all the $a_i = \mathsf{comPed}(\pi_i, x_{i,\pi_i})$, and check them with $\mathbf{a}$. The

erased random elements would help to check the encryptions of zeroes, what we do not want, since the equivocability property will exploit that.

Properties. Let us briefly check the security properties, which are formally proven in Appendix C. First, because of the perfectly hiding property of the Pedersen commitment, unless some information is leaked about the $x_{i,\delta}[j]$'s, no information is leaked about the π_i's. And granted the semantic security of the ElGamal encryption scheme, the former privacy is guaranteed. Since the Pedersen commitment is (computationally) binding, the a_i's cannot be opened in two ways, but only one pair (π_i, x_{i,π_i}) is possible. Let us now consider the new extended properties:

- (conditional) extractability is provided by the bit-by-bit encryption. With the decryption key sk, one can decrypt all the $b_{i,\delta}[j]$, and get the $x_{i,\delta}$ (unless the ciphertexts contain values different from 0 and 1, which will be one condition for extractability). Then, one can check, for $i = 1, \ldots, m$, whether $a_i = \mathsf{comPed}(0, x_{i,0})$ or $a_i = \mathsf{comPed}(1, x_{i,1})$, which provides π_i (unless none of the equalities is satisfied, which will be another condition for extractability).
- equivocability is possible using the Pedersen commitment trapdoor. Instead of taking a random x_{i,π_i} and then $x_{i,\overline{\pi}_i} = 0$, which specifies π_i as the committed bit, one takes a random $x_{i,0}$, computes $a_i = \mathsf{comPed}(0, x_{i,0})$, but also extracts $x_{i,1}$ so that $a_i = \mathsf{comPed}(1, x_{i,1})$ too (which is possible with the knowledge of discrete logarithm of y_i in basis g, the trapdoor). The rest of the commitment procedure remains the same, but now, one can open any bit-string for π, using the appropriate x_{i,π_i} and the corresponding random elements (the simulator did not erase).

5.2 The Associated Smooth Projective Hash Function

As noticed above, our new commitment scheme is conditionally extractable (one can recover the $x_{i,\delta}$'s, and then the committed value π), under the conditions that all the ElGamal ciphertexts encrypt either 0 or 1, and the a_i is a commitment of either 0 or 1, with random $x_{i,0}$ or $x_{i,1}$.

As before, one wants to make the two hash values (direct computation and the one from the projected key) be the same if the two parties use the same input π and perfectly independent if they use different inputs (smoothness). One furthermore wants to control that each a_i is actually a Pedersen commitment of π_i using the encrypted random x_{i,π_i}, and thus $g^{x_{i,\pi_i}} = a_i / y_i^{\pi_i}$: the extracted x_{i,π_i} is really the private key M related to a given public key g^M that is $a_i / y_i^{\pi_i}$ in our case. Using the same notations as in Section 4.1, we want to define a smooth hash system showing that, for all i, δ, j, $b_{i,\delta}[j] \in L_{(\mathbf{EG}^+,\rho),0\vee1}$ and, for all i, $B_{i,\pi_i} \in L_{(\mathbf{EG}^\times,\rho),(a_i/y_i{}^{\pi_i})}$, where $B_{i,\pi_i} = \prod_j b_{i,\pi_i}[j]$.

Combinations of these smooth hashes. Let C be the above commitment of π using randomness R as defined in Section 5.1. We now precise the language $L_{\rho,\pi}$, consisting informally of all the valid commitments "of good shape":

$$L_{\rho,\pi} = \left\{ C \;\middle|\; \begin{array}{ll} \exists R \text{ s. t. } C = \mathsf{com}_\rho(\pi, R) & \text{and } \forall i\ \forall j \quad b_{i,\pi_i}[j] \in L_{(\mathbf{EG}^+,\rho),0\vee1} \\ & \text{and } \forall i \quad B_{i,\pi_i} \in L_{(\mathbf{EG}^\times,\rho),a_i/y_i{}^{\pi_i}} \end{array} \right\}$$

The smooth hash system for this language relies on the smooth hash systems described previously, using the generic construction for conjunctions and disjunctions as described in Section 3. The precise definition of this language (which is constructed from conjunctions and disjunctions of simple languages) can be found in Appendix D, omitting the labels and replacing the Cramer-Shoup encryption $\mathbf{CS}^+$ by the ElGamal one $\mathbf{EG}^+$.

Properties: Uniformity and Independence. With a non-malleable variant of such a commitment and smooth hash function, it is possible to improve the establishment of a secure channel between two players, from the one presented Section 4.3. More precisely, two parties can agree on a common key if they both share a common (low entropy) password π. However, a more involved protocol than the one proposed in Section 4.3 is needed to achieve all the required properties of a password-authenticated key exchange protocol, as it will be explained in Section 5.3 and proven in Appendix E.

Nevertheless, there may seem to be a leakage of information because of the language that depends on the input π: the projected key hp seems to contain some information about π, that can be used

in another execution by an adversary. Hence the independence and uniformity notions presented Section 3.4, which ensure that hp does not contain any information about π. Proofs of these properties can be found in Appendix D.

Estimation of the Complexity. Globally, each operation (commitment, projected key, hashing and projected hashing) requires $\mathcal{O}(mn)$ exponentiations in G, with small constants (at most 16).

5.3 UC-Secure PAKE with Adaptive Security

The primitive presented above, but using the Cramer-Shoup encryption scheme (as described in Section D) is a non-malleable conditionally extractable and equivocable commitment. We now sketch how to use this new primitive in order to construct the first efficient adaptively-secure password-authenticated key exchange protocol in the UC framework with erasures. For lack of space, all the details can be found in Appendix E. The passwords are not known at the beginning of the simulation: $\mathcal{S}$ will manage to correct the errors (thanks to the equivocability) but without erasures there would remain clues on how the computations were held, which would give indications on the passwords used.

Our protocol is based on that of Gennaro and Lindell [22]. At a high level, the players in the KOY/GL protocol exchange CCA-secure encryptions of the password, under the public-key found in the common reference string, which are essentially commitments of the password. Then, they compute the session key by combining smooth projective hashes of the two password/ciphertext pairs. The security of this protocol relies on the properties of smoothness and pseudo-randomness of the smooth projective hash function. But as noted by Canetti *et al* in [13], the KOY/GL protocol is not known to achieve UC security: the main issue is that the ideal-model simulator must be able to extract the password used by the adversary before playing, which is impossible if the simulator is the initiator (on behalf of the client), leading to such situation in which the simulator is stuck with an incorrect ciphertext and will not be able to predict the value of the session key.

To overcome this problem, the authors of [13] made the client send a pre-flow which also contains an encryption of the password. The server then sends its own encryption, and finally the client sends another encryption, as well as a zero-knowledge proof showing that both ciphertexts are consistent and encrypt the same password. This time the simulator, playing as the client or the server, is able to use the correct password, recovered from the encrypted value sent earlier by the other party. The pre-flow is never used in the remaining of the protocol, hence the simulator can send a fake one, and simulate the zero-knowledge proof.

Unfortunately, the modification above does not seem to work when dealing with adaptive adversaries, which is the case in which we are interested. This is because the simulator cannot correctly open the commitment when the adversary corrupts the client after the pre-flow has been sent. A similar remark applies to the case in which the server gets corrupted after sending its first message. As a result, in addition to being extractable, the commitment scheme also needs to be equivocable for the simulator to be able to provide a consistent view to the adversary. Since the use of the equivocable and extractable commitment schemes also seems to solve the problem of proving the original Gennaro-Lindell protocol secure in the UC model, we opted to use that protocol as the starting point of our protocol.

These remarks are indeed enough (along with minor modifications) to obtain adaptive security. Thus, our solution essentially consists in using our non-malleable extractable and equivocable commitment scheme in the Gennaro-Lindell protocol when computing the first two flows. As presented in the previous subsections, extractability may be conditional: We include this condition in the language of the smooth hash function (note that the projected keys sent do not leak any information about the password). Additional technical modifications were also needed to make things work and can be found in Appendix E.

Acknowlegments

This work was supported in part by the French ANR-07-SESU-008-01 PAMPA Project, and the European ECRYPT Project.

References

1. M. Abdalla and D. Pointcheval. A scalable password-based group key exchange protocol in the standard model. In X. Lai and K. Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 332–347. Springer, Dec. 2006.
2. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 119–135. Springer, May 2001.
3. B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin. Secure computation without authentication. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377. Springer, Aug. 2005.
4. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, Oct. 1997.
5. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, May 2000.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.
7. S. M. Bellovin and M. 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.
8. J.-M. Bohli, M. I. Gonzalez Vasco, and R. Steinwandt. Password-authenticated constant-round group key establishment with a common reference string. Cryptology ePrint Archive, Report 2006/214, 2006. `http://eprint.iacr.org/`.
9. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Jan. 2003.
10. V. Boyko, P. D. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, May 2000.
11. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.
12. R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Aug. 2001.
13. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, May 2005.
14. R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, Apr. / May 2002.
15. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, Aug. 1998.
16. R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Apr. / May 2002.
17. I. Damgård and J. B. Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 581–596. Springer, Aug. 2002.
18. C. Delerablée and D. Pointcheval. Dynamic fully anonymous short group signatures. In P. Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06*, volume 4341 of *LNCS*, pages 193–210. Springer, Sept. 2006.
19. G. Di Crescenzo, J. Katz, R. Ostrovsky, and A. Smith. Efficient and non-interactive non-malleable commitment. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 40–59. Springer, May 2001.
20. Y. Dodis and J. Katz. Chosen-ciphertext security of multiple encryption. In J. Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 188–209. Springer, Feb. 2005.
21. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Aug. 1985.
22. R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, May 2003. `http://eprint.iacr.org/2003/032.ps.gz`.
23. O. Goldreich and Y. Lindell. Session-key generation using human passwords only. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 408–432. Springer, Aug. 2001. `http://eprint.iacr.org/2000/057`.
24. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In A. Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.
25. O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991.
26. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

27. Y. T. Kalai. Smooth projective hashing and two-message oblivious transfer. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 78–95. Springer, May 2005.
28. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 475–494. Springer, May 2001.
29. S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 465–485. Springer, May / June 2006.
30. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *12th SODA*, pages 448–457. ACM-SIAM, Jan. 2001.
31. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, May 1999.
32. V. Shoup. ISO 18033-2: An emerging standard for public-key encryption. `http://shoup.net/iso/std6.pdf`, Dec. 2004. Final Committee Draft.

A Formal Definitions for Smooth Projective Hash Functions

As defined in [22], a family of smooth projective hash functions, for a language $L_{\mathsf{pk},aux} \subset X$, onto the set G, based on a labeled encryption scheme with public key pk or on a commitment scheme with public parameters pk consists of four algorithms and is denoted by $\mathbf{HASH}(\mathsf{pk}) = (\mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$. Note that X is the range of either the encryption or commitment algorithm.

The probabilistic key-generation algorithm produces hash keys via $\mathsf{hk} \xleftarrow{\$} \mathsf{HashKG}(\mathsf{pk}, aux)$. The key projection algorithm produces projected hash keys via $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c)$, where c is either a ciphertext or a commitment in X. The hashing algorithm Hash computes, on $c \in X$, the hash value $g = \mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, c) \in G$, using the hash key hk. Finally, the projected hashing algorithm $\mathsf{ProjHash}$ computes, on $c \in X$, the hash value $g = \mathsf{ProjHash}(\mathsf{hp}; \mathsf{pk}, aux, c; w) \in G$, using the projected hash key hp and a witness w of the fact that $c \in L_{\mathsf{pk},aux}$.

We now recall the three properties of a smooth hash system.

Correctness. Let $c \in L_{\mathsf{pk},aux}$ and w a witness of this membership. Then, for all hash keys and projected hash keys $\mathsf{hk} \xleftarrow{\$} \mathsf{HashKG}(\mathsf{pk}, aux)$ and $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c)$, then $\mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, c) = \mathsf{ProjHash}(\mathsf{hp}; \mathsf{pk}, aux, c; w)$.

Smoothness. For every c which is *not* in $L_{\mathsf{pk},aux}$, the hash value $g = \mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, c)$ is statistically close to uniform and independent of the values hp, pk, aux and c: for uniformly-chosen hash key hk, the two distributions are statistically indistinguishable:

$$\{\mathsf{pk}, aux, c, \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c), g = \mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, c)\}$$
$$\{\mathsf{pk}, aux, c, \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c), g \xleftarrow{\$} G\}$$

Pseudorandomness. If $c \in L_{\mathsf{pk},aux}$, then without a witness w of this membership, the hash value $g = \mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, c)\}$ is computationally indistinguishable from random: for uniformly-chosen hash key hk, the following two distributions are computationally indistinguishable:

$$\{\mathsf{pk}, aux, c, \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c), g = \mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, c)\}$$
$$\{\mathsf{pk}, aux, c, \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c), g \xleftarrow{\$} G\}$$

Let $L_{\mathsf{pk},aux}$ be such that it is hard to distinguish a random element in $L_{\mathsf{pk},aux}$ from a random element not in $L_{\mathsf{pk},aux}$. Gennaro and Lindell formalized the latter property by showing in [22] that the two following experiments are indistinguishable:

`Expt-Hash`(D): Let D be a adversary that is given access to two oracles: Ω and `Hash`. The first oracle receives an empty input and returns $x \in L_{\mathsf{pk},aux}$ chosen according to the distribution of $L_{\mathsf{pk},aux}$. The Hash oracle receives an input x. If x was not previously outputted by Ω, it outputs nothing. Otherwise, it chooses a key hk and returns the pair $(\mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, x), \mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, x))$. The output of the experiment is whatever M outputs.

$\texttt{Expt-Unif}(D)$: This experiment is defined exactly as above except that the $\texttt{Hash}$ oracle is replaced by the following $\texttt{Unif}$ oracle. On input x, if x was not previously outputted by Ω, it outputs nothing. Otherwise, it chooses a key hk and a random element g and returns the pair $(\mathsf{ProjKG}(\mathsf{hk};\mathsf{pk},aux,x),g)$. The output of the experiment is whatever M outputs.

In the case where the language $L_{\mathsf{pk},aux}$ is associated with a labeled encryption scheme, we rename the oracle Ω to $\texttt{Enc}$. In the case where the language $L_{\mathsf{pk},aux}$ is associated with a commitment scheme, we rename the oracle Ω to $\texttt{Commit}$.

B Our Smooth Projective Hash Functions: Proofs of Section 3

In this appendix, we prove the properties of the smooth projective hash functions on conjunctions and disjunctions of languages.

B.1 Disjunction

We first deal with $L = L_1 \cup L_2$, and study the additional information provided by the projected key, which contains hp_1 and hp_2, but also $\mathsf{hp}_\Delta = \mathsf{Hash}_1(\mathsf{hk}_1;\rho,aux,x) \oplus \mathsf{Hash}_2(\mathsf{hk}_2;\rho,aux,x)$. Since both SHS_1 and SHS_2 are smooth projective hash functions, the pseudo-randomness property for each of them (or even the smoothness, if x does not lie in one of the languages) guarantees that the pairs $(\mathsf{hp}_i,\mathsf{Hash}_i(\mathsf{hk}_i;\rho,aux,x))$ are (statistically or computationally) indistinguishable from (hp_i,g_i). As a consequence, one easily gets that the tuple $(\mathsf{hp}_1,\mathsf{hp}_2,\mathsf{Hash}_1(\mathsf{hk}_1;\rho,aux,x),\mathsf{Hash}_2(\mathsf{hk}_2;\rho,aux,x))$ is (statistically or computationally) indistinguishable from $(\mathsf{hp}_1,\mathsf{hp}_2,g_1,g_2)$, where g_1 and g_2 are independent. This *a fortiori* implies that the tuple $(\mathsf{hp}_1,\mathsf{hp}_2,\mathsf{Hash}_1(\mathsf{hk}_1;\rho,aux,x) \oplus \mathsf{Hash}_2(\mathsf{hk}_2;\rho,aux,x))$ is (statistically or computationally) indistinguishable from $(\mathsf{hp}_1,\mathsf{hp}_2,g)$: the element hp_Δ does not provide any additional information.

Efficient Hashing from Key. Given any element $x \in X$ and a key hk, it is possible to efficiently compute $\mathsf{Hash}_L(\mathsf{hk};\rho,aux,x)$.

Proof. This follows from the efficient hashings of the two underlying smooth projective hash functions, and namely SHS_1, since $\mathsf{Hash}_L(\mathsf{hk};\rho,aux,x) = \mathsf{Hash}_1(\mathsf{hk}_1;\rho,aux,x)$.

Efficient Hashing from Projected Key. Given an element $x \in L$, a witness w of this membership, and the projected key $\mathsf{hp} = \mathsf{ProjKG}_L(\mathsf{hk};\rho,aux,x)$, it is possible to efficiently compute the projected hash value $\mathsf{ProjHash}_L(\mathsf{hp};\rho,aux,x,w) = \mathsf{Hash}_L(\mathsf{hk};\rho,aux,x)$.

Proof. If $x \in L_1$, then,

$$\begin{aligned}\mathsf{Hash}_L(\mathsf{hk};\rho,aux,x) &= \mathsf{Hash}_1(\mathsf{hk}_1;\rho,aux,x)\\ &= \mathsf{ProjHash}_1(\mathsf{hp}_1;\rho,aux,x,w) = \mathsf{ProjHash}_L(\mathsf{hp};\rho,aux,x,w),\end{aligned}$$

which can be computed efficiently since SHS_1 is a smooth projective hash function. If $x \in L_2$, then,

$$\begin{aligned}\mathsf{Hash}_L(\mathsf{hk};\rho,aux,x) &= \mathsf{Hash}_1(\mathsf{hk}_1;\rho,aux,x)\\ &= \mathsf{ProjHash}_1(\mathsf{hp}_1;\rho,aux,x,w) = \mathsf{hp}_\Delta \ominus \mathsf{ProjHash}_2(\mathsf{hp}_2;\rho,aux,x,w),\end{aligned}$$

which can be computed efficiently since SHS_2 is a smooth projective hash function.

Smoothness. For each element $x \in X \setminus L$, $\mathsf{Hash}_L(\mathsf{hk};\rho,aux,x)$ is uniformly distributed, given the projected key.

Proof. Consider $x \notin L$. Then, $x \notin L_1$ and $x \notin L_2$. If $\mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2)$ is a random key, and $\mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2, \mathsf{hp}_\Delta)$ the corresponding projected key, then the above analysis showed the statistical indistinguishability between $(\mathsf{hp}_1, \mathsf{hp}_2, \mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, x), \mathsf{Hash}_2(\mathsf{hk}_2; \rho, aux, x))$ and $(\mathsf{hp}_1, \mathsf{hp}_2, g_1, g_2)$, where g_1 and g_2 are random and independent elements. This means the statistical indistinguishability between the tuples $(\mathsf{hp}_1, \mathsf{hp}_2, \mathsf{hp}_\Delta, \mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, x))$ and $(\mathsf{hp}_1, \mathsf{hp}_2, g_1, g_2)$. As a consequence, the tuple $(\mathsf{hp}, \mathsf{Hash}_L(\mathsf{hk}; \rho, aux, x))$ is statistically indistinguishable from (hp, g), which is the definition of the smoothness for the new system.

Pseudo-Randomness. For each element $x \in L$, the value $\mathsf{Hash}_L(\mathsf{hk}; \rho, aux, x)$ is computationally indistinguishable from uniform, given the projected key.

Proof. Exactly the same analysis as in the previous paragraph can be done, but with computational indistinguishability when $x \in L_1$ or $x \in L_2$. Hence one gets that the tuple $(\mathsf{hp}, \mathsf{Hash}_L(\mathsf{hk}; \rho, aux, x))$ is computationally indistinguishable from (hp, g), which is the definition of the pseudo-randomness for the new system.

B.2 Conjunction

The case of $L = L_1 \cap L_2$, can be dealt as above: $(\mathsf{hp}_1, \mathsf{hp}_2, \mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, x) \oplus \mathsf{Hash}_2(\mathsf{hk}_2; \rho, aux, x))$ is statistically (if $x \notin L_1$ or $x \notin L_2$) or computationally (if $x \in L_1 \cap L_2$) indistinguishable from the tuple $(\mathsf{hp}_1, \mathsf{hp}_2, g)$.

B.3 Preservation of the Uniformity and Independence Properties.

If the two underlying smooth hash systems verify 1-uniformity (resp. 2-uniformity), then the smooth hash system for conjunction or disjunction verifies these properties. If the two underlying smooth hash systems verify 1-independence (resp. 2-independence), then the smooth hash system for conjunction verifies these properties. We insist on the fact that independence does not propagate to disjunction, since the hash value (that needs both aux and x) is included in the projected key.

Proof. We only prove the result for 1-uniformity for disjunction (the proof is the same in the other cases —excepted independence for disjunction, where the result does not hold). Then, if (ρ, aux) are the parameters of the languages and x and x' belong to L, $D_{1,\rho,aux,x} \approx D_{1,\rho,aux,x'}$ and $D_{2,\rho,aux,x} \approx D_{2,\rho,aux,x'}$. Due to the form of hp, this ensures that the first two element of hp are indistinguishable. We now have to consider the third part. Without loss of generality, we can suppose that $x \in L_1$. Then, due to the pseudo-randomness of the first smooth-hash, the value $\mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, x)$ is computationally indistinguishable from uniform. This is at least the same for the value $\mathsf{Hash}_2(\mathsf{hk}_2; \rho, aux, x)$. Since the projected keys depend on independent random values and languages, both parts of the $\oplus$ are independent: the value $\mathsf{Hash}_L(\mathsf{hk}_L; \rho, aux, x)$ is then indistinguishable from uniform. As a result, $D_{\rho,aux,x} \approx D_{\rho,aux,x'}$.

C Proofs for Commitment in Section 5

Extractability. The extraction key is sk, the ElGamal decryption key (or the Cramer-Shoup one in the non-malleable version of the primitive, also used for the ElGamal decryption). First, the simulator tries to decrypt all the $b_{i,\delta}[j]$ into $x_{i,\delta}[j]$ and aborts if one of them is not either 0 or 1. It then builds up the $x_{i,\delta}$, and checks whether $a_i = g^{x_{i,0}}$ or $a_i = g^{x_{i,1}} y_i$, which makes it recover π_i (unless none or both are satisfied). This extraction only fails in three cases:

- First, if the ciphertexts do not encrypt 0 or 1;
- Second, if a_i satisfies none of the equalities;
- Third, if a_i satisfies both equalities.

The two first reasons will be excluded in the language L, while the third one would break the binding property of the Pedersen commitment, which leads to the discrete logarithm of some y_i in base g. It thus happens only with negligible probability.

Equivocability. Note that, with the knowledge of the discrete logarithms of $(y_1, \ldots, y_m)$, one is able to compute, for all $i \in \{1, \ldots, m\}$, both $x_{i,0}$ and $x_{i,1}$. The equivocation thus consists in computing, for all i, an encryption of both $x_{i,0}$ and $x_{i,1}$ (and not that of 0). Provided one does not erase any random, this allows one to change its mind and open each commitment on any π_i (0 or 1).

We now prove that committing to all the bit-string π in a unique commitment does not change the view of an adversary. The proof is based on a Left-or-Right hybrid argument, see [4]. We suppose the existence of an oracle answering with either $\mathcal{E}(g^x)$ or $\mathcal{E}(g^0)$ when provided with g^x (and g^0 is implicitly given). We then define hybrid games in which the encryptions for $x_{i,\pi_i}[j]$ in the commitment are computed with the help of this oracle. The first hybrid game, in which the oracle always encrypts g^0, is equivalent to the real computation, where only one bit-string is committed (perfectly binding). Similarly, the last one, in which it always encrypts g^x, is equivalent to the simulation, where all the bit-strings are committed (equivocable).

$$\left|\begin{array}{c} G_{1a} \\ 1 \\ 1 \\ \vdots \\ 1 \end{array}\right| \quad \left|\begin{array}{c} G_{1-b} = G_{2-a} \\ g^{x_{1,\overline{\pi}_1}[1]} \\ 1 \\ \vdots \\ 1 \end{array}\right| \quad \left|\begin{array}{c} G_{2-b} = G_{3-a} \\ g^{x_{1,\overline{\pi}_1}[1]} \\ g^{x_{1,\overline{\pi}_1}[2]} \\ \vdots \\ 1 \end{array}\right| \quad \cdots \quad \left|\begin{array}{c} G_{mn-b} \\ g^{x_{1,\overline{\pi}_1}[1]} \\ g^{x_{1,\overline{\pi}_1}[2]} \\ \vdots \\ g^{x_{m,\overline{\pi}_m}[n]} \end{array}\right|$$

$$\longleftrightarrow \quad \mathtt{Adv}_{\mathtt{cpa}}(\mathcal{E}) \qquad \longleftrightarrow \quad \mathtt{Adv}_{\mathtt{cpa}}(\mathcal{E}) \qquad \longleftrightarrow \quad \mathtt{Adv}_{\mathtt{cpa}}(\mathcal{E})$$

Hiding. This property simply follows from the equivocability of the commitment.

Binding. First suppose that we do not know the discrete logarithms of $(y_1, \ldots, y_m)$ in base g and that the adversary has managed to send a commitment that can be opened on π and on π', then, for some bit $i = 1, \ldots, m$, $\pi_i \neq \pi'_i$, and the adversary is able to give us $x_i \neq x'_i$ such that $g^{x_i} y^{\pi_i} = g^{x'_i} y^{\pi'_i}$. This event thus boils down to breaking the discrete logarithm problem, which happens only with negligible probability. This means that if equivocability is not used, the commitment is (computationally) binding, under the discrete logarithm problem.

Now, since a real commitment (with a zero encryption) and an equivocable commitment (with no zero encryption) are indistinguishable for an adversary, the view of equivocable commitments does not help the adversary to break the binding property, or otherwise it would break the IND-CPA property of the underlying encryption scheme.

D A Non-Malleable Conditionally-Extractable Equivocable Commitment

In this section, we show how to enhance the previous commitment schemes as described in Section 5 with non-malleability: briefly, one simply needs to extend the ElGamal commitment to the labeled Cramer-Shoup one together with one-time signatures. As before, if b is a bit, we denote its complement by $\overline{b}$ (i.e., $\overline{b} = 1 - b$). We furthermore denote by $x[i]$ the i^{th} bit of the bit-string x.

D.1 Non-Malleability

Non-malleability is a usual requirement for encryption schemes or commitments [22]. We thus now aim at achieving this property. We thus use labeled Cramer-Shoup encryption instead of ElGamal, and we add a one-time signature. Using the results of Dodis and Katz [20] for chosen-ciphertext security of multiple encryption, one can easily show that the chosen-ciphertext security (and thus non-malleability) of the combined encryption scheme used to compute a ciphertext vector $(b_1, \ldots, b_m)$

follows trivially from the chosen-ciphertext security of the underlying labeled Cramer-Shoup scheme and strong unforgeability of the one-time signature scheme used to link all the ciphertexts together.

More precisely, if M is defined as before, and ℓ is a label, the commitment $\mathsf{comCS}^{\ell}_{\mathsf{pk}}(M)$ is obtained as follows. First, the user generates a key pair $(\mathrm{VK}, \mathrm{SK})$ for a one-time signature scheme. Then, it computes the following values, with $\ell' = \ell \circ \mathrm{VK}$:

$$\forall i \qquad b_i = \mathsf{CS}^{+\,\ell'}_{\mathsf{pk}}(M_i \cdot 2^{i-1}, r_i) = (u_{1,i} = g_1^{r_i}, u_{2,i} = g_2^{r_i}, e_i = h^{r_i} g^{M_i \cdot 2^{i-1}}, v_i = (cd^{\theta_i})^{r_i}).$$

Defining $\mathbf{b} = (b_1, \ldots, b_m)$, it computes $\sigma = \mathsf{Sign}(\mathrm{SK}, \mathbf{b})$. The final commitment is then defined as $\mathsf{comCS}^{\ell'}_{\mathsf{pk}}(M) = (\mathbf{b}, \mathrm{VK}, \sigma)$. One can obtain, from any b_i, an ElGamal encryption of $M_i \cdot 2^{i-1}$: $B_i = \mathsf{EG}^{+}_{\mathsf{pk}}(M_i \cdot 2^{i-1}, r_i) = (u_{1,i} = g_1^{r_i}, e_i = h^{r_i} g^{M_i \cdot 2^{i-1}})$. The homomorphic property of the encryption scheme allows to obtain $B = \mathsf{EG}^{\times}_{\mathsf{pk}}(g^M, \sum r_i) = \mathsf{EG}^{+}_{\mathsf{pk}}(M, \sum r_i) = (\prod u_{1,i}, \prod e_i) = \prod B_i$.

In order to define the smooth projective hashing associated with this commitment scheme, we recall the family of smooth projective hashing functions for the underlying labeled Cramer-Shoup encryption scheme, as defined in [22].

Let $X' = G^4$ and $L' = L_{(\mathbf{CS}^+,\rho),(\ell,M)}$ be the language of the elements C such that C is a valid Cramer-Shoup encryption of M under the label ℓ (aux is defined as (ℓ, M)). Under the DDH assumption, this is a hard subset membership problem. Denoting by $C = \mathsf{CS}^{+\,\ell}_{\mathsf{pk}}(M, r) = (u_1, u_2, e, v)$, the associated smooth hash system is the following:

$$\begin{aligned}
\mathsf{HashKG}((\mathbf{CS}^+, \rho), (\ell, M)) &= \mathsf{hk} = (\gamma_1, \gamma_2, \gamma_3, \gamma_4) \xleftarrow{\$} \mathbb{Z}_q \times \mathbb{Z}_q \times \mathbb{Z}_q \times \mathbb{Z}_q \\
\mathsf{ProjKG}(\mathsf{hk}; (\mathbf{CS}^+, \rho), (\ell, M), C) &= \mathsf{hp} = (g_1)^{\gamma_1} (g_2)^{\gamma_2} (h)^{\gamma_3} (cd^{\theta})^{\gamma_4} \\
\mathsf{Hash}(\mathsf{hk}; (\mathbf{CS}^+, \rho), (\ell, M), C) &= (u_1)^{\gamma_1} (u_2)^{\gamma_2} (eg^M)^{\gamma_3} (v)^{\gamma_4} \\
\mathsf{ProjHash}(\mathsf{hp}; (\mathbf{CS}^+, \rho), (\ell, M), C; r) &= (\mathsf{hp})^r
\end{aligned}$$

From these definitions, we consider the language: $L_{(\mathbf{CS}^+,\rho),(\ell,0\vee1)} = L_{(\mathbf{CS}^+,\rho),(\ell,0)} \cup L_{(\mathbf{CS}^+,\rho),(\ell,1)}$, where $L_{(\mathbf{CS}^+,\rho),(\ell,0)} = \{C \mid \exists r\ C = \mathsf{CS}^{+\,\ell}_{\mathsf{pk}}(0, r)\}$ and $L_{(\mathbf{CS}^+,\rho),(\ell,1)} = \{C \mid \exists r\ C = \mathsf{CS}^{+\,\ell}_{\mathsf{pk}}(1, r)\}$, and we want to define a smooth hash system showing that

$$\forall i \quad b_i \in L_{(\mathbf{CS}^+,\rho),(\ell',0\vee1)} \qquad \text{and} \qquad B = \prod B_i \in L_{(\mathbf{EG}^\times,\rho),g^M}.$$

Note that, for all i, the first membership also implies that $B_i \in L_{(\mathbf{EG}^+,\rho),0\vee1}$, since the ElGamal ciphertexts B_i are extracted from the corresponding Cramer-Shoup ciphertexts b_i: It is thus enough to check the validity of the latter membership to get extractability, which means that one can extract the committed private key M, associated to the public key g^M. The signature has of course to be verified too, but this can be publicly performed, from VK.

D.2 Equivocability

We here describe our commitment for completeness, but note that it is very similar to the one described in Section 5.

Description of the Commitment. Our scheme uses the labeled Cramer-Shoup public-key encryption scheme and a one-time signature scheme, to achieve non-malleability as explained in the previous section. More precisely, the specific example given in this section relies on the labeled version of the Cramer-Shoup encryption scheme [15], used for each bit of the bit-string, seen as an extension of the homomorphic ElGamal encryption [21].

Let the input of the committing algorithm be a bit-string $\pi = \sum_{i=1}^{m} \pi_i \cdot 2^{i-1}$ and a label ℓ. The algorithm works as follows:

- For $i = 1, \ldots, m$, it chooses a random value $x_{i,\pi_i} = \sum_{j=1}^{n} x_{i,\pi_i}[j] \cdot 2^{j-1}$ and sets $x_{i,\overline{\pi}_i} = 0$. It also generates a key pair $(\mathrm{VK}, \mathrm{SK})$ for a one-time signature scheme.
- For $i = 1, \ldots, m$, it commits to π_i, using the random x_{i,π_i}: $a_i = \mathsf{comPed}(\pi_i, x_{i,\pi_i}) = g^{x_{i,\pi_i}} y_i^{\pi_i}$ and defining $\mathbf{a} = (a_1, \ldots, a_m)$.

– For $i = 1, \ldots, m$, using $\ell_i = \ell \circ \mathrm{VK} \circ \mathbf{a} \circ i$, it computes the Cramer-Shoup commitments (see the previous section) of $x_{i,\delta}$, for $\delta = 0, 1$:

$$(\mathbf{b}_{i,\delta} = (b_{i,\delta}[j])_j, \mathrm{VK}, \sigma_{i,\delta}) = \mathsf{comCS}^{\ell_i}_{\mathsf{pk}}(x_{i,\delta}), \text{ where } b_{i,\delta}[j] = \mathsf{CS}^{+\,\ell_i}_{\mathsf{pk}}(x_{i,\delta}[j] \cdot 2^{j-1}, r_{i,\delta}[j]).$$

The computation of the $b_{i,\delta}[j]$ implicitly defines $B_{i,\delta}[j] = \mathsf{EG}^{+}_{\mathsf{pk}}(x_{i,\delta}[j] \cdot 2^{j-1}, r_{i,\delta}[j])$, from which one can directly extract an encryption $B_{i,\delta}$ of $x_{i,\delta}$: $B_{i,\delta} = \prod_j B_{i,\delta}[j] = \mathsf{EG}^{+}_{\mathsf{pk}}(x_{i,\delta}, r_{i,\delta})$, where $r_{i,\delta}$ is the sum of the random coins $r_{i,\delta}[j]$.

The entire random string for this commitment is (where n is the bit-length of the prime order q of the group G) $R = (x_{1,\pi_1}, (r_{1,0}[1], r_{1,1}[1], \ldots, r_{1,0}[n], r_{1,1}[n]), \ldots, x_{m,\pi_m}, (r_{m,0}[1], \ldots, r_{m,1}[n]), \mathrm{SK})$. From which, all the values $r_{i,\overline{\pi}_i}[j]$ can be erased, letting the opening data (witness of the committed value) become $\mathsf{w} = (x_{1,\pi_1}, (r_{1,\pi_1}[1], \ldots, r_{1,\pi_1}[n]), \ldots, x_{m,\pi_m}, (r_{m,\pi_m}[1], \ldots, r_{m,\pi_m}[n]))$.

The output of the committing algorithm, of the bit-string π, with the label ℓ, and using the random R, is $\mathsf{com}_\rho(\ell, \pi; R) = (\ell, \mathbf{a}, \mathbf{b}, \mathrm{VK}, \sigma)$, where $\mathbf{a} = (a_i = \mathsf{comPed}(\pi_i, x_{i,\pi_i}))_i$, $\mathbf{b} = (b_{i,\delta}[j] = \mathsf{CS}^{+\,\ell_i}_{\mathsf{pk}}(x_{i,\delta}[j] \cdot 2^{j-1}, r_{i,\delta}[j]))_{i,\delta,j}$, and $\sigma = (\sigma_{i,\delta} = \mathsf{Sign}(\mathrm{SK}, (b_{i,\delta}[j])_j))_{i,\delta}$.

Properties. This commitment is opened as in Section 5 and the proofs given in Section C still hold.

Furthermore, let us show now that the view of equivocable commitments does not help the adversary to build a valid but non-extractable commitment (it could be open in any way, and thus extraction leads to many possibilities) due to the CCA property of the encryption. As in Appendix C, we use a Left-or-Right argument, see [4]. The "left" oracle, which always provides the player with an encryption $\mathcal{E}(g^0)$, is equivalent to the game where no equivocable commitments are available, and the "right" oracle, which always provides him with $\mathcal{E}(g^x)$, is equivalent to the game where the commitments are equivocable. Then, if the adversary was more likely to build a valid but non-extractable commitment in the latter case than in the former one, one could construct a distinguisher to the Left-or-Right oracles. However, contrarily to the proof in Appendix C, a decryption oracle is required to check whether the commitment is valid but non-extractable (whereas in Appendix C the adversary breaks the binding property with two different opening values).

As a consequence, producing valid but non-extractable commitments is not easier when equivocable commitments are provided to the adversary, than when no equivocable commitments are provided, under the IND-CCA security of the encryption scheme. Furthermore, a valid but non-extractable commitment, with a decryption oracle leads to two different opening values for the commitment, and thus to an attack against the binding property, which relies on the discrete logarithm problem.

The additional property is non-malleability, and it is guaranteed by the labeled Cramer-Shoup encryption scheme (IND-CCA) and the one-time signature, as already explained [20]. More precisely, for the results of Dodis and Katz [20] for chosen-ciphertext security of multiple encryption, one can easily show that the chosen-ciphertext security of the combined encryption scheme used to compute ciphertext vector $\mathbf{b}$ follows trivially from the chosen-ciphertext security of the underlying labeled Cramer-Shoup scheme and strong unforgeability of the one-time signature scheme used to link all the ciphertexts together.

D.3 The Associated Smooth Projective Hash Function

As noticed above, our new commitment scheme is conditionally extractable (one can recover the $x_{i,\delta}$'s, and then π), under the conditions that all the Cramer-Shoup ciphertexts encrypt either 0 or 1, and the a_i is a commitment of either 0 or 1, with random $x_{i,0}$ or $x_{i,1}$.

Since we want to apply it later to the password-based setting, we want to make the two hash values (direct computation and the one from the projected key) to be the same if the two parties use the same password π, but perfectly independent if they use different passwords: using their password $\pi = \sum_{i=1}^{m} \pi_i \cdot 2^{i-1}$, one furthermore wants to ensure that each a_i is actually a Pedersen commitment of π_i using the encrypted random x_{i,π_i}, and thus $g^{x_{i,\pi_i}} = a_i / y_i^{\pi_i}$: the extracted x_{i,π_i} is really the private key M related to a given public key g^M that is $a_i / y_i^{\pi_i}$ in our case. Using the same notations as in

Section D.1, we want to define a smooth hash system showing that, for all i, δ, j, $b_{i,\delta}[j] \in L_{(\mathbf{CS}^+,\rho),(\ell_i,0\vee1)}$ and, for all i, $B_{i,\pi_i} \in L_{(\mathbf{EG}^\times,\rho),(a_i/y_i{}^{\pi_i})}$, where $B_{i,\pi_i} = \prod_j B_{i,\pi_i}[j]$. As before, note that, for all i, δ, j, the first membership also implies that $B_{i,\delta}[j] \in L_{(\mathbf{EG}^+,\rho),0\vee1}$ since this value is extracted from the corresponding value $b_{i,\delta}[j]$: It is thus enough to check the validity of the latter.

Combinations of these smooth hashes. Let C be the above commitment of π using label ℓ and randomness R as defined in Section D.2. We now precise the language $L_{\rho,(\ell,\pi)}$, consisting informally of all the valid commitments "of good shape":

$$L_{\rho,(\ell,\pi)} = \left\{ C \;\middle|\; \begin{array}{l} \exists R \text{ such that } C = \mathsf{com}_\rho(\ell, \pi, R) \\ \quad \text{and } \forall i\ \forall \delta\ \forall j \quad b_{i,\delta}[j] \in L_{(\mathbf{CS}^+,\rho),(\ell_i,0\vee1)} \\ \quad \text{and } \forall i \quad B_{i,\pi_i} \in L_{(\mathbf{EG}^\times,\rho),a_i/y_i{}^{\pi_i}} \end{array} \right\}$$

The smooth hash system for this language relies on the smooth hash systems described in the previous subsection, using the generic construction for conjunctions and disjunctions as described in Section 3. More precisely, adding the values $B_{i,\delta}$ easily computed from the values $b_{i,\delta}[j]$, the commitment can be converted into a tuple of the following form (we omit the signature part, since it has to be verified before applying any hashing computation):

$$\begin{array}{c}(\ell, a_1, \ldots, a_m, b_{1,0}[1], b_{1,1}[1], \ldots, b_{1,0}[n], b_{1,1}[n], \ldots, b_{m,0}[1], b_{m,1}[1], \ldots, b_{m,0}[n], b_{m,1}[n], \\ B_{1,0}, B_{1,1}, \ldots, B_{m,0}, B_{m,1}) \in \{0,1\}^* \times G^m \times (G^4)^{2mn} \times (G^2)^{2m}.\end{array}$$

We denote by $L^{i,\delta,j}_{(\mathbf{CS}^+,\rho),(\ell_i,0\vee1)}$ the language that restricts the value $b_{i,\delta}[j]$ to be a valid Cramer-Shoup encryption of either 0 or 1:

$$\begin{array}{c}\underbrace{\{0,1\}^*}_{\ell} \times \underbrace{G^m}_{\mathbf{a}} \times \underbrace{(G^4)^{2n}}_{b_{1,*}[*]} \times \ldots \times (\underbrace{G^4}_{b_{i,0}[1]} \times \underbrace{G^4}_{b_{i,1}[1]} \times \ldots \times \underbrace{L_{(\mathbf{CS}^+,\rho),(\ell_i,0\vee1)}}_{b_{i,\delta}[j]} \times \ldots \times \underbrace{G^4}_{b_{i,0}[n]} \times \underbrace{G^4}_{b_{i,1}[n]}) \\ \times \ldots \times \underbrace{(G^4)^{2n}}_{b_{1,m}[*]} \times \underbrace{(G^2)^{2m}}_{B_{*,*}}\end{array}$$

and by $L^i_{(\mathbf{EG}^\times,\rho)}$ the language that restricts the B_{i,π_i} ElGamal ciphertexts:

$$\text{if } \pi_i = 0, L^i_{(\mathbf{EG}^\times,\rho)} = \underbrace{\{0,1\}^*}_{\ell} \times \underbrace{G^m}_{\mathbf{a}} \times \underbrace{(G^4)^{2mn}}_{b_{*,*}[*]} \times \underbrace{(G^2)^2}_{B_{1,*}} \times \ldots \times (\underbrace{L_{(\mathbf{EG}^\times,\rho),a_i}}_{B_{i,0}} \times \underbrace{G^2}_{B_{i,1}}) \times \ldots \times \underbrace{(G^2)^2}_{B_{m,*}}$$

$$\text{if } \pi_i = 1, L^i_{(\mathbf{EG}^\times,\rho)} = \underbrace{\{0,1\}^*}_{\ell} \times \underbrace{G^m}_{\mathbf{a}} \times \underbrace{(G^4)^{2mn}}_{b_{*,*}[*]} \times \underbrace{(G^2)^2}_{B_{1,*}} \times \ldots \times (\underbrace{G^2}_{B_{i,0}} \times \underbrace{L_{(\mathbf{EG}^\times,\rho),a_i/y_i}}_{B_{i,1}}) \times \ldots \times \underbrace{(G^2)^2}_{B_{m,*}}.$$

Then, our language $L_{\rho,(\ell,\pi)}$ is the conjunction of all these languages, where the $L^{i,\delta,j}_{(\mathbf{CS}^+,\rho),(\ell_i,0\vee1)}$'s are disjunctions:

$$L_{\rho,(\ell,\pi)} = \left(\bigcap_{i,\delta,j} L^{i,\delta,j}_{(\mathbf{CS}^+,\rho),(\ell_i,0\vee1)} \right) \cap \left(\bigcap_i L^i_{(\mathbf{EG}^\times,\rho)} \right).$$

Properties: Uniformity and Independence. With such a commitment and smooth hash function, it is possible to improve the establishment of a secure channel between two players, from the one presented Section 4.3. More precisely, two parties can agree on a common key if they both share a common (low entropy) password π. However, a more involved protocol than the one proposed in Section 4.3 is needed to achieve all the required properties of a password-authenticated key exchange protocol, as it will be explained and proven in the next appendix.

Nevertheless, there may seem to be a leakage of information because of the language that depends on the password π: the projected key hp seems to contain some information about π, that can be used in another execution by an adversary. Hence the independence and uniformity notions presented Section 3.4, which ensure that hp does not contain any information about π:

- for the languages $L^{i}_{(\mathbf{EG}^{\times},\rho)}$, the smooth hash functions satisfy the *2-independence* property, since the projected key for a language $L_{(\mathbf{EG}^{\times},\rho),M}$ depends on the public key (and thus ρ) only. One thus generates one key for each pair $(B_{i,0}, B_{i,1})$ only, and uses the correct ciphertext according to actual/wanted π_i when evaluating the hash value. But this distinction on π_i appears in the computation of the hash value only, that is pseudo-random. The projected key is totally independent of π_i.
- for the languages $L^{i,\delta,j}_{(\mathbf{CS}^{+},\rho),(\ell_i,0\vee 1)}$, the smooth hash functions satisfy the *2-uniformity* property only, but not *2-independence*. Therefore, the projected key and (aux, ρ) are statistically independent, but the former depends, in its computation, of the latter. If we want the equivocability property for the commitment (as needed in the next section), we have to include all the pairs $(b_{i,0}[j], b_{i,1}[j])$, and not only the $b_{i,\pi_i}[j]$, so that we can open later in any way. Because of 2-uniformity instead of 2-independence, we need a key for each element of the pair, and not only one has above.

Estimation of the Complexity. Globally, each operation (commitment, projected key, hashing and projected hashing) requires $\mathcal{O}(mn)$ exponentiations in G, with small constants (at most 16).

Let us first consider the commitment operation, on a m-bit secret π, over a n-bit group G. One has first to make m Pedersen commitments of one bit (m exponentiations) and then $2mn$ additive Cramer-Shoup encryptions (2 exponentiations and 2 multi-exponentiations each, and thus approximately the cost of $8mn$ exponentiations).

About the smooth hash function, the hash key generation just consists in generating random elements in $\mathbb{Z}_q$: 8 for each Cramer-Shoup ciphertext, in order to show that they encrypt either 0 or 1, and 2 for each pair of ElGamal ciphertexts, then globally $2m(8n+1)$ random elements in $\mathbb{Z}_q$. The projected keys need exponentiations: $m(4n+1)$ multi-exponentiations, and $4mn$ hash evaluations (one multi-exponentiation each). Then, globally, the cost of $m(8n+1)$ exponentiations in G is required for the projected key. Finally, the hash computations are essentially the same using the hash key or the projected key, since for the sub-functions, the former consists of one multi-exponentiation and the latter consists of 1 exponentiation. They both cost $m(4n+1)$ exponentiations, after the multiplications needed to compute the B_{i,π_i}, which are negligible.

E A New Adaptively-Secure PAKE Protocol in the UC Framework

E.1 Password-Based Key Exchange and Universal Composability

The Password-Based Key Exchange Functionality. In this section, we present the password-based key-exchange functionality $\mathcal{F}_{pwKE}$ (see Figure 1) first described in [13]. The main idea behind this functionality is as follows: If neither party is corrupted, then they both end up with the same uniformly-distributed session key, and the adversary learns nothing about it (except that it was indeed generated). However, if one party is corrupted, or if the adversary successfully guessed the player's password (the session is then marked as `compromised`), then it is granted the right to fully determine its session key. Note that as soon as a party is corrupted, the adversary learns its key: There is in fact nothing lost by allowing it to determine the key.

In addition, the players become aware of a failed attempt of the adversary at guessing a password. This is modeled by marking the session as `interrupted`. In this case, the two players are given independently-chosen random keys.

A session that is nor `compromised` nor `interrupted` is called `fresh`. In such a case, the two parties receive the same, uniformly distributed session key.

Finally notice that the functionality is not in charge of providing the password(s) to the participants. The passwords are chosen by the environment which then hands them to the parties as inputs. This

guarantees security even in the case where two honest players execute the protocol with two different passwords: This models, for instance, the case where a user mistypes its password. It also implies that the security is preserved for all password distributions (not necessarily the uniform one) and in all situations where the password is used in different protocols. Also note that allowing the environment to choose the passwords guarantees forward secrecy.

The functionality $\mathcal{F}_{pwKE}$ is parameterized by a security parameter k. It interacts with an adversary $\mathcal{S}$ and a set of parties $P_1, \ldots, P_n$ via the following queries:

- **Upon receiving a query (`NewSession`, $sid, P_i, P_j, pw, role$) from party P_i:**
  Send (`NewSession`, $sid, P_i, P_j, role$) to $\mathcal{S}$. If this is the first `NewSession` query, or if this is the second `NewSession` query and there is a record (P_j, P_i, pw'), then record (P_i, P_j, pw) and mark this record `fresh`.
- **Upon receiving a query (`TestPwd`, sid, P_i, pw') from the adversary S:**
  If there is a record of the form (P_i, P_j, pw) which is `fresh`, then do: If $pw = pw'$, mark the record `compromised` and reply to $\mathcal{S}$ with "correct guess". If $pw \neq pw'$, mark the record `interrupted` and reply with "wrong guess".
- **Upon receiving a query (`NewKey`, sid, P_i, sk) from the adversary S:**
  If there is a record of the form (P_i, P_j, pw), and this is the first `NewKey` query for P_i, then:
  - If this record is `compromised`, or either P_i or P_j is corrupted, then output (sid, sk) to player P_i.
  - If this record is `fresh`, and there is a record (P_j, P_i, pw') with $pw' = pw$, and a key sk' was sent to P_j, and (P_j, P_i, pw) was `fresh` at the time, then output (sid, sk') to P_i.
  - In any other case, pick a new random key sk' of length k and send (sid, sk') to P_i.

  Either way, mark the record (P_i, P_j, pw) as `completed`.

Fig. 1. The password-based key-exchange functionality $\mathcal{F}_{pwKE}$

The KOY/GL Protocol. The starting point of our protocol is the password-based key exchange protocol of Katz, Ostrovsky and Yung [28], generalized by Gennaro and Lindell in [22]. At a high level, the players in the KOY/GL protocol exchange CCA-secure encryptions of the password, under the public-key found in the common reference string, which is essentially a commitment of the password. Then, they compute the session key by combining smooth projective hashes of the two password/ciphertext pairs. More precisely, each player chooses a hashing key for a smooth projective hash function and sends the corresponding projected key to the other player. Each player can thus compute the output of its own hash function with the help of the hashing key, and the output of the other one using the projected key and its knowledge of the randomness that was used to generate the ciphertext of the password. All the flows generated by a party are linked together with a one-time signature, generated in the last flow, but which public key is included in the label of the CCA-secure encryption of the password.

To understand informally why this protocol is secure, first consider the case in which the adversary plays a passive role. In this case, the pseudo-randomness property of the smooth hash function ensures that the value of the session key will be computationally indistinguishable from uniform since the adversary does not know the randomness that was used to encrypt the password. Now imagine the case in which the adversary provides the user with an encryption of the wrong password. In this case, the security of the protocol will rely on the smoothness of the hash functions, which ensures that the session key will be random and independent of all former communication. Thus, in order to be successful, the adversary has to generate the encryption of the correct password. To do so, the adversary could try to copy or modify existing ciphertexts. Since the encryption scheme is CCA-secure, and thus non-malleable, modifying is not really a possibility. Copying does not help either since either the label used for encryption will not match (making the session key look random due to the smoothness property) or the signature will be invalid (in the case where the adversary changes the projection keys without changing the label and hence the verification key). As a result, the only successful strategy left

for the adversary is essentially to guess the password and perform the trivial online dictionary attack, as desired.

Extending the protocol to the UC Framework (Static Case). As noted by Canetti *et al* in [13], the KOY/GL protocol is not known to achieve UC security: the main issue is that the ideal-model simulator must be able to extract the password used by the adversary. One could think that, since the simulator has control over the common reference string, it knows all private keys corresponding to the public keys and can thus decrypt all ciphertexts sent by the adversary and recover its password.

But indeed, this doesn't not seem to be sufficient. In the case where the adversary begins to play (*i.e.* it impersonates the client), everything works well: The simulator decrypts the ciphertext generated by the adversary and can thus recover the password it has used. If the guess of the adversary is incorrect (that is, the password is the wrong one), then the smoothness of the hash functions leads to random independent session keys. Otherwise, if the guess is correct, the execution can continue as an honest one would do (the simulator has learned which password to use).

However, let's now suppose that the simulator has to start the game, on behalf of the client. Here, the simulator needs to send an encryption of the password before having seen anything coming from the adversary. As described above, it recovers the password used by the adversary as soon as the latter has sent its value, but this is too late. If it turns out that the guess of the adversary is incorrect, there is no problem thanks to the smoothness, but otherwise, the simulator is stuck with an incorrect ciphertext and will not be able to predict the value of the session key.

To overcome this problem, the authors of [13] made the client send a pre-flow which also contains an encryption of the password. The server then sends its own encryption, and finally the client sends another encryption (this time the simulator is able to use the correct password, recovered from the value sent by the adversary), as well as a zero-knowledge proof claiming that both ciphertexts are consistent and encrypt the same password. The first flow is never used in the remaining of the protocol. This solves the problem since on the one hand, the simulator is of course able to give a valid proof of a false statement, and on the other hand, the first flow will never be used afterwards.

E.2 Description of the Protocol

As explained above, Canetti *et al.* [13] proposed a simple variant of the Gennaro-Lindell methodology [22] that is provably secure in the UC framework. Though quite efficient, their protocol is not known to be secure against adaptive adversaries. The only one PAKE adaptively-secure in the UC framework was proposed by Barak *et al.* [3] using general techniques from multi-party computation. It thus leads to quite inefficient schemes.

In the following, we use our non-malleable conditionally-extractable and equivocable commitment scheme with an associated smooth projective hash function family, in order to build an efficient PAKE scheme, adaptively-secure in the common reference string model under standard complexity assumptions, in the UC framework.

The complete description of the protocol can be found in Figure 3, whereas an informal sketch is presented in Figure 2. We use the index I for a value related to the client Alice, and J for a value related to the server Bob.

Correctness. In an honest execution of the protocol, if the players share the same password ($\mathsf{pw}_\mathrm{I} = \mathsf{pw}_\mathrm{J} = \mathsf{pw}$), it is easy to verify that both players will terminate by accepting and computing the same values for the session key, equal to $\mathsf{Hash}(\mathsf{hk}_\mathrm{I}; \mathsf{pw}, \ell_\mathrm{J}, \mathsf{com}_\mathrm{J}) + \mathsf{Hash}(\mathsf{hk}_\mathrm{J}; \mathsf{pw}, \ell_\mathrm{I}, \mathsf{com}_\mathrm{I})$.

Security. The intuition behind the security of our protocol is quite simple and builds on that of Gennaro-Lindell protocol. The key point in order to achieve adaptive security is the use of the commitment, which allows for extraction and equivocation at any moment, thus not requiring the simulator to be aware of future corruptions. The following theorem, which full proof will be given in Appendix E.3, states that the protocol is UC-secure. The ideal functionality $\mathcal{F}_{pwKE}$ has been presented in Figure 1 and described in Appendix E.1. Since we use the joint state version of the UC theorem, we implicitly consider the multi-session extension of this functionality. In particular, note that the passwords of the

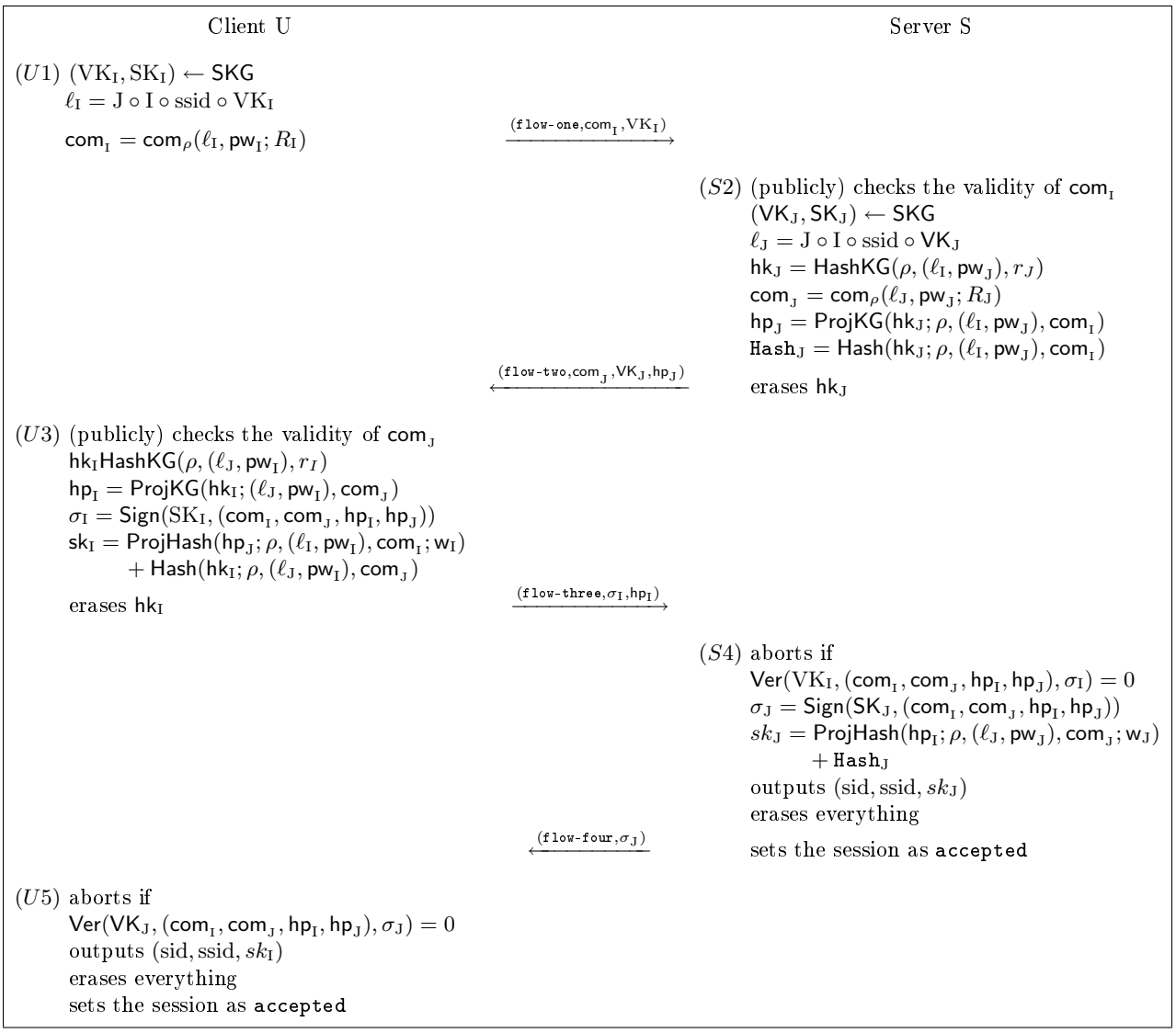

Fig. 2. Description of the protocol for players $(P_{\mathrm{I}}, \mathrm{ssid})$, with index I and password pw_{I} and $(P_{\mathrm{J}}, \mathrm{ssid})$, with index J and password pw_{J}. At the end of rounds 1 and 2, the players will erase the part of the random values R_{I} and R_{J} used in the commitment which is not needed in the following rounds, keeping only w_{I} and w_{J}.

players depend on the session considered. For sake of simplicity, we denote them by pw_{I} and pw_{J}, but one should implicitly understand $\mathsf{pw}_{\mathrm{I,ssid}}$ and $\mathsf{pw}_{\mathrm{J,ssid}}$.

Theorem 1. *Let* com *be the non-malleable (conditionally) extractable and equivocable committing scheme described in Section D, $\mathcal{H}$ be a family of smooth hash functions with respect to this commitment, and $\mathcal{SIG}$ be a one-time signature scheme. Denote by $\widehat{\mathcal{F}}_{pwKE}$ the multi-session extension of the functionality $\mathcal{F}_{pwKE}$ of password-based key-exchange, and let $\mathcal{F}_{CRS}$ be the ideal functionality that provides a common reference string $(G, \mathsf{pk}, (y_1, \ldots, y_m), \mathsf{Extract})$ to all parties, where G is a cyclic group, $y_1, \ldots, y_m$ random elements from this group,* pk *a public key for the Cramer-Shoup scheme and* $\mathsf{Extract}$ *a randomness extractor. Then, the above protocol securely realizes $\widehat{\mathcal{F}}_{pwKE}$ in the $\mathcal{F}_{CRS}$-hybrid model, in the presence of adaptive adversaries.*

E.3 Proof of Theorem 1

Sketch of Proof. In order to prove Theorem 1, we need to construct, for any real-world adversary $\mathcal{A}$ (interacting with real parties running the protocol), an ideal-world adversary $\mathcal{S}$ (interacting with

Common reference string. A tuple $(G, \mathsf{pk}, (y_1, \ldots, y_m), \mathsf{Extract})$, where G is a cyclic group, $y_1, \ldots, y_m$ random elements from this group, pk a public key for the Cramer-Shoup scheme, and $\mathsf{Extract}$ a randomness extractor.

Protocol steps.

1. When P_I is activated with input $(\texttt{NewSession}, \mathrm{sid}, \mathrm{ssid}, \mathrm{I}, \mathrm{J}, \mathsf{pw}_\mathrm{I}, \texttt{role})$, we face two cases: If $\texttt{role} = \texttt{server}$, it does nothing. If $\texttt{role} = \texttt{client}$, it uses SKG to generate a key pair $(\mathrm{VK}_\mathrm{I}, \mathrm{SK}_\mathrm{I})$ for a one-time signature scheme, sets the (public) label $\ell_\mathrm{I} = \mathrm{J} \circ \mathrm{I} \circ \mathrm{ssid} \circ \mathrm{VK}_\mathrm{I}$, computes $\mathsf{com}_\mathrm{I} = \mathsf{com}_\rho(\ell_\mathrm{I}, \mathsf{pw}_\mathrm{I}; R_\mathrm{I})$ and sends the message $(\texttt{flow-one}, \mathsf{com}_\mathrm{I}, \mathrm{VK}_\mathrm{I})$ to P_J. From this point on, assume that P_I is a party activated with input $(\texttt{NewSession}, \mathrm{sid}, \mathrm{ssid}, \mathrm{I}, \mathrm{J}, \mathsf{pw}_\mathrm{I}, \texttt{client})$ and that P_J is a party activated with input $(\texttt{NewSession}, \mathrm{sid}, \mathrm{ssid}, \mathrm{I}, \mathrm{J}, \mathsf{pw}_\mathrm{J}, \texttt{server})$. Recall that P_I erases nearly all the randoms (in R_I) used in the computation of com_I (see Section D). More precisely, it only keeps from R_I the values present in the witness w_I that will be used in the computation of the smooth hash function.
2. When P_J receives a message $(\texttt{flow-one}, \mathsf{com}_\mathrm{I}, \mathrm{VK}_\mathrm{I})$, it (publicly) checks that com_I is well constructed. Otherwise, it aborts. Then it uses SKG to generate a key pair $(\mathsf{VK}_\mathrm{J}, \mathsf{SK}_\mathrm{J})$ for a one-time signature scheme, and HashKG to generate a key hk_J for the smooth projective hash function family $\mathcal{H}$ with respect to ρ. It sets the (public) label $\ell_\mathrm{J} = \mathrm{J} \circ \mathrm{I} \circ \mathrm{ssid} \circ \mathsf{VK}_\mathrm{J}$ and computes the projection $\mathsf{hp}_\mathrm{J} = \mathsf{ProjKG}(\mathsf{hk}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{I})$. Next it computes $\mathsf{com}_\mathrm{J} = \mathsf{com}_\rho(\ell_\mathrm{J}, \mathsf{pw}_\mathrm{J}; R_\mathrm{J})$ and sends the message $(\texttt{flow-two}, \mathsf{com}_\mathrm{J}, \mathrm{VK}_\mathrm{J}, \mathsf{hp}_\mathrm{J})$ to P_I. It finally computes $\mathsf{Hash}_J = \mathsf{Hash}(\mathsf{hk}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{I})$ and erases hk_J and the values of R_J that are not present in the witness w_J.
3. When P_I receives a message $(\texttt{flow-two}, \mathsf{com}_\mathrm{J}, \mathrm{VK}_\mathrm{J}, \mathsf{hp}_\mathrm{J})$, it (publicly) checks that com_J is well constructed. Otherwise, it aborts. Then it uses HashKG to generate a key hk_I for the smooth projective hash function family $\mathcal{H}$ with respect to pk and computes the projection $\mathsf{hp}_\mathrm{I} = \mathsf{ProjKG}(\mathsf{hk}_\mathrm{I}; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{I}), \mathsf{com}_\mathrm{J})$. Next it computes $\sigma_\mathrm{I} = \mathsf{Sign}(\mathrm{SK}_\mathrm{I}, (\mathsf{com}_\mathrm{I}, \mathsf{com}_\mathrm{J}, \mathsf{hp}_\mathrm{I}, \mathsf{hp}_\mathrm{J}))$ and sends the message $(\texttt{flow-three}, \sigma_\mathrm{I}, \mathsf{hp}_\mathrm{I})$ to P_J. It computes the session key $sk_\mathrm{I} = \mathsf{ProjHash}(\mathsf{hp}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{I}), \mathsf{com}_\mathrm{I}; \mathsf{w}_\mathrm{I}) + \mathsf{Hash}(\mathsf{hk}_\mathrm{I}; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{I}), \mathsf{com}_\mathrm{J})$, and erases all private data except pw_I and sk_I, keeping all public data in memory.
4. When P_J receives a message $(\texttt{flow-three}, \sigma_\mathrm{I}, \mathsf{hp}_\mathrm{I})$, it checks that $\mathsf{Ver}(\mathrm{VK}_\mathrm{I}, (\mathsf{com}_\mathrm{I}, \mathsf{com}_\mathrm{J}, \mathsf{hp}_\mathrm{J}), \sigma_\mathrm{I}) = 1$. If not, it aborts the session outputting nothing. Otherwise, it computes $\sigma_\mathrm{J} = \mathsf{Sign}(\mathrm{SK}_\mathrm{J}, (\mathsf{com}_\mathrm{I}, \mathsf{com}_\mathrm{J}, \mathsf{hp}_\mathrm{I}, \mathsf{hp}_\mathrm{J}))$ and sends the message $(\texttt{flow-four}, \sigma_\mathrm{J})$ to P_I. Then, it computes the session key $\mathsf{sk}_\mathrm{J} = \mathsf{Hash}_\mathrm{J} + \mathsf{ProjHash}_\mathrm{I}(\mathsf{hp}_\mathrm{I}; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{J}; \mathsf{w}_\mathrm{J})$, outputs $(\mathrm{sid}, \mathrm{ssid}, \mathsf{sk}_\mathrm{J})$, sets the session as $\texttt{accepted}$, which means in particular that it erases everything except pw_J and sk_J and then terminates (i.e., publishes the session key).
5. When P_I receives a message $(\texttt{flow-four}, \sigma_\mathrm{J})$, it checks that $\mathsf{Ver}(\mathsf{VK}_\mathrm{J}, (\mathsf{com}_\mathrm{I}, \mathsf{com}_\mathrm{J}, \mathsf{hp}_\mathrm{I}, \mathsf{hp}_\mathrm{J}), \sigma_\mathrm{J}) = 1$. If not, it aborts the session outputting nothing. Otherwise, it terminates the session (i.e., publishes the session key).

Fig. 3. Description of the protocol for two players Alice and Bob. Alice is the client P_I, with index I and password pw_I, and Bob is the server P_J, with index J and password pw_J.

dummy parties and the functionality $\mathcal{F}_{pwKE}$) such that no environment $\mathcal{Z}$ can distinguish between an execution with $\mathcal{A}$ in the real world and $\mathcal{S}$ in the ideal world with non-negligible probability.

We first describe two hybrid queries that are going to be used in the games. The `GoodPwd` query checks whether the password of some player is the one we have in mind or not. The `SamePwd` query checks if the players share the same password, without disclosing it. In some games the simulator has actually access to the players. In such a case, a `GoodPwd` (or a `SamePwd`) query can be easily implemented by letting the simulator look at the passwords owned by the oracles. When the players are entirely simulated, $\mathcal{S}$ will replace the queries above with `TestPwd` and `NewKey` queries.

We say that a flow is *oracle-generated* if it was sent by an honest player and arrives without any alteration to the player it was meant to. We say it is *non-oracle-generated* otherwise, that is either if it was sent by an honest player and modified by the adversary, or if it was sent by a corrupted player or a player impersonated by the adversary (more generally denoted by *attacked* player, that is, a player whose password is known to the adversary).

We incrementally define a sequence of games starting from the one describing a real execution of the protocol and ending up with game $\boldsymbol{G}_8$ which we prove to be indistinguishable with respect to the ideal experiment. For the sake of clarity, we will use the following notation: the client is Alice (hence She), the server is Bob (hence He), and we use It for the adversary $\mathcal{A}$ and the simulator $\mathcal{S}$.

– $\boldsymbol{G}_0$ is the real game.

- From $\boldsymbol{G}_1$, $\mathcal{S}$ is allowed to program the CRS.
- From $\boldsymbol{G}_2$, $\mathcal{S}$ always extracts the password committed to by the adversary and aborts when the extraction fails due to the commitment being valid for two or more passwords. This would lead to an attack against the binding property of the commitment scheme.
- From $\boldsymbol{G}_3$, $\mathcal{S}$ simulates all the commitments and makes them equivocable granted the simulated CRS. The commitment remains binding and hiding for the environment and the adversary, which follows from the CCA2-property of the encryption. As a side note, the knowledge of the passwords is not necessary anymore for the simulation of the committing step.
- From $\boldsymbol{G}_4$, $\mathcal{S}$ simulates the honest client without using her password anymore (except for the hybrid queries). She is given a random key in $(U3)$ in the case where `flow-two` was oracle-generated and the server was not corrupted. If the server was corrupted, $\mathcal{S}$ recovers his password pw_J and makes a call to the `GoodPwd` functionality for Alice. If it is incorrect, Alice is also given a random key, but if it is correct, the key is computed honestly using that password. If no password is recovered, Alice is also given a random key. Alice then aborts in $(U5)$ if the signature of the server is invalid. If the server was corrupted before $(U5)$, $\mathcal{S}$ recovers his password and does exactly the same as previously described. This is indistinguishable from the former game due to the pseudo-randomness of the hash function.
- From $\boldsymbol{G}_5$, in the case where `flow-two` was not oracle-generated, $\mathcal{S}$ extracts pw_J from com_J and proceeds as described in $\boldsymbol{G}_4$: it asks a `GoodPwd` query for Alice and provides her with either a random value or a value computed honestly for sk_I. Similarly, if no password is recovered, Alice is given a random key. Alice aborts in $(U5)$ if the signature of the server is invalid. A corruption of the server before $(U5)$ is dealt with as in $\boldsymbol{G}_4$. This is indistinguishable from the former game due to the smoothness of the hash function.
- From $\boldsymbol{G}_6$, $\mathcal{S}$ simulates the server without using his password anymore. It aborts if the signature received from the client is invalid. Otherwise, the server is given a random key in $(S4)$ in the case where `flow-one` was oracle-generated and the client was not corrupted. A corruption of the client before $(S4)$ is dealt with as in $\boldsymbol{G}_4$: $\mathcal{S}$ asks a `GoodPwd` query for Bob and provides him with either a random value or a value computed honestly for sk_J (if no password is recovered, Bob is given a random key). This is indistinguishable from the former game due to the pseudo-randomness of the hash function.
- From $\boldsymbol{G}_7$, in the case where `flow-one` was not oracle-generated, $\mathcal{S}$ extracts pw_I from com_I and proceeds as described in $\boldsymbol{G}_4$: it asks a `GoodPwd` query for Bob and provides him with either a random value or a value computed honestly for sk_J (if no password is recovered, Bob is given a random key). This is indistinguishable from the former game due to the smoothness of the hash function.
- Finally, the hybrid queries are replaced by the real ones in $\boldsymbol{G}_8$, which is shown to be indistinguishable to the ideal-world experiment.

Description of the simulator The description of our simulator is based on that of [13]. When initialized with security parameter k, the simulator first runs the key-generation algorithm of the encryption scheme $\mathcal{E}$, thus obtaining a pair $(\mathsf{sk}, \mathsf{pk})$. It also chooses at random m elements $(y_1, \ldots, y_m)$ in G and a randomness extractor $\mathsf{Extract}$. It then initializes the real-world adversary $\mathcal{A}$, giving it $(\mathsf{pk}, (y_1, \ldots, y_m), \mathsf{Extract})$ as common reference string.

From this moment on, the simulator interacts with the environment $\mathcal{Z}$, the functionality $\mathcal{F}_{pwKE}$ and its subroutine $\mathcal{A}$. For the most part, this interaction is implemented by the simulator $\mathcal{S}$ just following the protocol on behalf of all the honest players. The main difference between the simulated players and the real honest players is that $\mathcal{S}$ does not engage on a particular password on their behalf. However, if $\mathcal{A}$ modifies a `flow-one` or a `flow-two` message that is delivered to player P in session ssid, then $\mathcal{S}$ decrypts that ciphertext (using sk) and uses the recovered message pw in a `TestPwd` query to the functionality. If this is a correct guess, then $\mathcal{S}$ uses this password on behalf of P, and proceeds with the simulation. More details follow.

CORRUPTIONS. Since we consider adaptive corruptions, that can occur at any moment during the execution of the protocol, our simulator, given the password of a player, needs to be able to provide the adversary with an internal state consistent with any data already sent (without the knowledge of the player's password at that time). To handle such corruptions, the key point relies in the equivocable property of our commitment. More precisely, instead of committing to a particular password, the simulator commits to *all* passwords, being able in the end to open to any of them. In a nutshell, committing to all passwords means to simulate the commitment in such a way that encryptions of all x_i, corresponding to all pw_i, are sent instead of encryptions of 0 (see the "equivocability" part of Section D for details). Recall that the values hk and hp do not depend on the password, so that it does not engage the player on any of them.

SESSION INITIALIZATION. When receiving a message $(\texttt{NewSession}, \text{sid}, \text{ssid}, \text{I}, \text{J}, \texttt{role})$ from $\mathcal{F}_{pwKE}$, $\mathcal{S}$ starts simulating a new session of the protocol for party P_I, peer P_J, session identifier ssid, and common reference string $(\mathsf{pk}, (y_1, \ldots, y_m), \mathsf{Extract})$. We denote this session by $(P_\text{I}, \text{ssid})$. If $\mathsf{role} = \texttt{client}$, then $\mathcal{S}$ generates a `flow-one` message by committing to all the passwords, choosing a key pair $(\text{SK}_\text{I}, \text{VK}_\text{I})$ for a one-time signature scheme. It gives this message to $\mathcal{A}$ on behalf of $(P_\text{I}, \text{ssid})$.

If $(P_\text{I}, \text{ssid})$ gets corrupted at this stage, then $\mathcal{S}$ recovers his password pw_I and is able to open his commitment in such a way that it is a commitment on pw_I. It can thus provide $\mathcal{A}$ with consistent data.

PROTOCOL STEPS. Assume that $\mathcal{A}$ sends a message m to an active session of some party. If this message is formatted differently from what is expected by the session, then $\mathcal{S}$ aborts that session and notifies $\mathcal{A}$. Otherwise, we have the following cases (where we denote a party in the client role as P_I and a party in the server role as P_J):

1. Assume that the session $(P_\text{J}, \text{ssid})$ receives a message $m = (\texttt{flow-one}, \mathsf{com}_\text{I}, \text{VK}_\text{I})$. Then, P_J is necessarily a server and m is the first message received by P_J. If com_I is not equal to any commitment that was generated by $\mathcal{S}$ for a `flow-one` message, $\mathcal{S}$ uses its secret key sk to decrypt the ciphertext and obtain pw_I or nothing. Obtaining nothing is considered similar to an invalid password below due to the construction of the smooth hash function related to the commitment. When the extraction succeeds, because of the binding property, only one pw_I is possible (on-line dictionary attack), then $\mathcal{S}$ makes a call $(\texttt{TestPwd}, \text{sid}, \text{ssid}, \text{J}, \mathsf{pw}_\text{I})$ to the functionality. If this is a correct guess, then $\mathcal{S}$ sets the password of this server session to pw_I, otherwise, this is an invalid password. In both cases, $\mathcal{S}$ produces a commitment com_J on all the passwords (it makes use of the equivocable property), chooses a key pair $(\text{SK}_\text{J}, \text{VK}_\text{J})$ for a one-time signature scheme, runs the key generation algorithms of the smooth hash function on com_I to produce $(\mathsf{hk}_\text{J}, \mathsf{hp}_\text{J})$ and sends the `flow-two` message $(\mathsf{com}_\text{J}, \mathsf{VK}_\text{J}, \mathsf{hp}_\text{J})$ to $\mathcal{A}$ on behalf of $(P_\text{J}, \text{ssid})$.

   If the sender $(P_\text{I}, \text{ssid})$ of this message, or if $(P_\text{J}, \text{ssid})$ gets corrupted at the end of this step, $\mathcal{S}$ handles this corruption just as when this player gets corrupted at the end of the initialization step. Note in addition that $\mathcal{S}$ is able to compute and give to $\mathcal{A}$ a correct value for Hash_J, the projected key being independent of the password (see discussion in Section D.3).
2. Assume that a session $(P_\text{I}; \text{ssid})$ receives a message $m = (\texttt{flow-two}, \mathsf{com}_\text{J}, \mathsf{VK}_\text{J}, \mathsf{hp}_\text{J})$. Then, P_I must be a client who sent a `flow-one` message and is now waiting for the response. We say that $(P_\text{J}, \text{ssid})$ is a peer session to $(P_\text{I}, \text{ssid}')$ if $\text{ssid} = \text{ssid}'$, if session $(P_\text{I}, \text{ssid})$ has peer P_J, session $(P_\text{J}, \text{ssid})$ has peer P_I, and these two sessions have opposite roles (`client` /`server`). If the pair $(\mathsf{com}_\text{J}, \mathsf{VK}_\text{J})$ is not equal to the pair $(\mathsf{com}_\text{J}, \mathsf{VK}_\text{J})$ that was generated by $\mathcal{S}$ for a `flow-one` message from peer session $(P_\text{J}, \text{ssid})$ (or if no such ciphertext was generated yet, or no such peer session exists) then $\mathcal{S}$ uses its secret key sk to compute pw_J, or nothing which is considered similar to an invalid password below (as before). Also note that $\mathcal{S}$ can recover pw_J if $(P_\text{J}, \text{ssid})$ has been corrupted after having sent its commitment. In case of recovery of pw_J, $\mathcal{S}$ then makes a call $(\texttt{TestPwd}, \text{sid}, \text{ssid}, \text{I}, \mathsf{pw}_\text{J})$ to the functionality. If this is a correct guess, $\mathcal{S}$ sets the password of this client session to pw_J, otherwise, this is an invalid password.

Then, it runs the key generation algorithms of the smooth hash function on $\mathsf{com}_{\mathrm{J}}$ to produce $(\mathsf{hk}_{\mathrm{I}}, \mathsf{hp}_{\mathrm{I}})$, as well as the signing algorithm with SK_{I} to compute σ_{I} and sends the `flow-three` message $(\sigma_{\mathrm{I}}, \mathsf{hp}_{\mathrm{I}})$ to $\mathcal{A}$ on behalf of $(P_{\mathrm{I}}, \mathrm{ssid})$.
Note that in the former case (correct password guess), $\mathcal{S}$ computes honestly the session key using password pw_{J}, without issuing it yet. Otherwise, P_{I} is provided with a key chosen at random.

If $(P_{\mathrm{J}}, \mathrm{ssid})$ gets corrupted at the end of this step, $\mathcal{S}$ handles this corruption just as when this player gets corrupted at the end of the previous step.
If it is $(P_{\mathrm{I}}, \mathrm{ssid})$ that gets corrupted, we face two cases. If a correct password guess occurred in this step, then $\mathcal{S}$ has computed everything honestly and can provide every value to $\mathcal{A}$ (recall that the projection key does not depend on the password). Otherwise, if $\mathcal{S}$ has set the session key at random, recall that it has not sent anything yet, so that the adversary totally ignores the values computed. $\mathcal{S}$ then recovers the password of $(P_{\mathrm{I}}, \mathrm{ssid})$ and is able to compute the data and give them to the adversary.

3. Assume that a session $(P_{\mathrm{J}}, \mathrm{ssid})$ receives a message $m = (\texttt{flow-three}, \sigma_{\mathrm{I}}, \mathsf{hp}_{\mathrm{I}})$. Then, $(P_{\mathrm{J}}, \mathrm{ssid})$ must be a server who sent a `flow-two` message and is now waiting for the response. $\mathcal{S}$ aborts if the signature σ_{I} is not valid. If `flow-one` was not oracle-generated, then $\mathcal{S}$ has extracted the password pw_{I} from the commitment (or failed to extract it). Similarly, if the peer session P_{I} (exists and) was corrupted sooner in the protocol, then $\mathcal{S}$ knows its password pw_{I} (which was in particular used in the commitment). In both cases, the simulator makes a call $(\texttt{TestPwd}, \mathrm{sid}, \mathrm{ssid}, \mathrm{J}, \mathsf{pw}_{\mathrm{I}})$ to the functionality to check the compatibility of the two passwords. In case of a correct answer, $\mathcal{S}$ sets the password of this server session to pw_{I}, and computes honestly the session key using password pw_{I}. Otherwise, P_{J} is provided with a key chosen at random.
Next, $\mathcal{S}$ runs signing algorithm with SK_{J} to compute σ_{J} and sends this `flow-four` message to $\mathcal{A}$ on behalf of $(P_{\mathrm{J}}, \mathrm{ssid})$.

   $\mathcal{S}$ handles a corruption of $(P_{\mathrm{I}}, \mathrm{ssid})$ just as it did at the end of the former step. And recall that no more corruption of $(P_{\mathrm{J}}, \mathrm{ssid})$ can occur since it claimed its session as "completed" and erased its data.
4. Assume that a session $(P_{\mathrm{I}}, \mathrm{ssid})$ receives a message $m = (\texttt{flow-four}, \sigma_{\mathrm{J}})$. $(P_{\mathrm{I}}, \mathrm{ssid})$ must then be a client who sent a `flow-three` message and is now waiting for the response. $\mathcal{S}$ aborts if the signature σ_{J} is not valid. If `flow-two` was not oracle-generated, then $\mathcal{S}$ has extracted the password pw_{J} from the commitment (or failed to). Similarly, if the peer session P_{J} (exists and) was corrupted sooner in the protocol, then $\mathcal{S}$ knows its password pw_{J} (which was in particular used in the commitment). In both cases, the simulator makes a call $(\texttt{TestPwd}, \mathrm{sid}, \mathrm{ssid}, \mathrm{J}, \mathsf{pw}_{\mathrm{J}})$ to the functionality to check the compatibility of the two passwords. In case of a correct answer, $\mathcal{S}$ sets the password of this client session to pw_{J} and computes honestly its session key. Otherwise, it sets its session key at random. Finally note that no corruption can occur at this stage.

If a session aborts or terminates, $\mathcal{S}$ reports it to $\mathcal{A}$. If the session terminates with a session key sk, then $\mathcal{S}$ makes a `NewKey` call to $\mathcal{F}_{pwKE}$, specifying the session key. But recall that unless the session is compromised or corrupted, $\mathcal{F}_{pwKE}$ will ignore the key specified by $\mathcal{S}$, and thus we do not have to bother with the key in these cases.

Description of the games We now provide the complete proof by a sequence of games. The detailed proof of some gaps are provided in Appendix F.

Game G_0: G_0 is the real game.

Game G_1: From this game on, we allow the simulator to program the common reference string, allowing it to know the trapdoors for extractability and equivocability.

Game G_2: This game is almost the same as the previous one. The only difference is that $\mathcal{S}$ always tries to extract the password committed to by the adversary (without taking advantage of the knowledge of this password for the moment) whenever the latter attempts to impersonate one of the parties.

We allow the simulator to abort whenever this extraction fails because the adversary has generated a commitment which is valid for two or more passwords. Due to the binding property of the commitment (see Section D), the probability that the adversary achieves this is negligible. Note that the extraction can also fail if the values sent were not encryptions of 0 or 1 but we do not abort in this case; For the moment we still assume that the simulator knows the passwords of the players. In the following games, when it will not have this knowledge anymore, we will show that the smooth hashes will be random so that this failure will have no bad consequences. This shows that $\boldsymbol{G}_2$ and $\boldsymbol{G}_1$ are indistinguishable.

Game $\boldsymbol{G}_3$: In this game, $\mathcal{S}$ still knows the passwords of both players, but it starts simulating the commitments by committing to all possible passwords (in order to be able to equivocate afterwards, see Section D for details). Note that since the commitment is hiding, this does not change the view of an environment, and that the commitment remains binding (even under access to equivocable commitments —see Section D and Appendix C). Also note that the generation of the projected keys for the smooth hash function (see Section D) is done without requiring the knowledge of the password. Hence, $\boldsymbol{G}_3$ and $\boldsymbol{G}_2$ are indistinguishable.

As a side note, we have just proven that the knowledge of the passwords is not necessary anymore for steps $(U1)$ and $(S2)$. If a player gets corrupted, the simulator recovers its password and is able to equivocate the commitment and thus provide the adversary with consistent data (since the projected key for the smooth hash function does not depend on the password committed).

Game $\boldsymbol{G}_4$: In this game, we suppose that `flow-one` was oracle-generated. We are now at beginning of round $(U3)$ and we want to simulate the (honest) client. We suppose that the simulator still knows the password of the server but not that of the client anymore. Let's first consider the case in which Alice received a `flow-two` which was oracle-generated. In such a case, the simulator chooses a random value $\mathsf{Hash}(\mathsf{hk}_\mathrm{I}; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{I}), \mathsf{com}_\mathrm{J})$. Then, if the server remains honest until $(S4)$, the simulator asks a `SamePwd` query to the functionality. If the answer is yes (that is $\mathsf{pw}_\mathrm{I} = \mathsf{pw}_\mathrm{J}$), it gives Bob the same random value for $\mathsf{ProjHash}(\mathsf{hp}_\mathrm{I}; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{J}; \mathsf{w}_\mathrm{J})$ and computes honestly $\mathsf{Hash}(\mathsf{hk}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{I})$, thus completely determining sk_J. Otherwise, it computes correctly the entire key. If both remain honest until $(U5)$, then, if they have the same password, $\mathcal{S}$ sets $\mathsf{ProjHash}(\mathsf{hp}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{I}), \mathsf{com}_\mathrm{I}; \mathsf{w}_\mathrm{I}) = \mathsf{Hash}(\mathsf{hk}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{I})$. Otherwise, $\mathcal{S}$ sets sk_I at random.

The description of the corruptions and the proof of the indistinguishability between $\boldsymbol{G}_4$ and $\boldsymbol{G}_3$ can be found in Appendix F.1 (it follows from the pseudo-randomness of the smooth hash function).

Game $\boldsymbol{G}_5$: In this game, we still suppose that `flow-one` was oracle-generated, but we now consider the case in which `flow-two` was non-oracle-generated. We are now at beginning of round $(U3)$ and we want to simulate the (honest) client. The simulator still knows the password of the server.

com_J, VK_J or hk_J has been generated by the adversary. If com_J is a replay, then the adversary could not modify VK_J, because of the label used for com_J, and then the signature verification will fail. com_J is thus necessarily a new commitment. Recall that from $\boldsymbol{G}_2$, the simulator extracts the password from the commitment of the server, with the help of the secret key. The extraction can fail with negligible probability if $\mathcal{S}$ recovers two different passwords (thanks to the binding property, see $\boldsymbol{G}_2$ and $\boldsymbol{G}_3$). In such a case, the simulator aborts the game. It can also happen that the extraction issues no password or that it fails if the values sent were not encryptions of 0 or 1.

Then, if it has recovered a password, the simulator asks a `GoodPwd` query to the functionality. If it is correct, then it computes honestly the session key of the client. Otherwise, or if it has not recovered any password in the extraction, it sets the value $\mathsf{Hash}(\mathsf{hk}_\mathrm{I}; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{I}), \mathsf{com}_\mathrm{J})$ at random (the entire key sk_I will be set in round $(U5)$). Note that the smooth hash value will necessarily be random by construction if the values sent in the commitment were not encryptions of 0 or 1.

The corruptions which can follow this step are dealt with as in the former game (recall that the projected keys sent do not depend on the password). Due to the smoothness of the hash function, this game is indistinguishable from the previous one (see Appendix F.2).

Game $\boldsymbol{G}_6$: In this game, we deal with the case in which all flows received by the client up to $(S4)$ were oracle-generated. We are now at beginning of round $(S4)$ and we want to simulate the (honest) server. We suppose that the simulator doesn't know any passwords anymore. Let's first consider the case in which `flow-three` was oracle-generated. Note that `flow-one` must have been oracle-generated too, otherwise the signature σ_1 would have been rejected.

Then, the simulator asks a `SamePwd` query to the functionality. If it is correct, it sets the value for $\mathsf{ProjHash}(\mathsf{hp}_\mathrm{I}; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{J}; \mathsf{w}_\mathrm{J})$ equal to $\mathsf{Hash}(\mathsf{hk}_\mathrm{I}; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{J})$, and it also sets the value for $\mathsf{Hash}(\mathsf{hk}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{I})$ at random. Otherwise, it sets these values at random.

In $(U5)$, if the passwords of Alice and Bob are the same and both remain honest, we set the value $\mathsf{ProjHash}(\mathsf{hp}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{I}), \mathsf{com}_\mathrm{I}; \mathsf{w}_\mathrm{I}) = \mathsf{Hash}(\mathsf{hk}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{I})$: sk_I and sk_J are thus equal, as required. Note that since $\mathsf{Hash}(\mathsf{hk}_\mathrm{I}; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{I}), \mathsf{com}_\mathrm{J})$ is already set at random since $\boldsymbol{G}_4$ or $\boldsymbol{G}_5$, all the keys are already random.

If their passwords are not the same but both remain honest, Alice will be given in $(U5)$ a key chosen independently at random: Here, $\mathsf{ProjHash}(\mathsf{hp}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{I}), \mathsf{com}_\mathrm{I}; \mathsf{w}_\mathrm{I})$ doesn't have to be programmed, since the keys do not have any reason to be identical. In this case, as in $\boldsymbol{G}_4$, the pseudo-randomness of the hash functions ensures the indistinguishability (see Appendix F.3 for the treatment of corruptions).

Game $\boldsymbol{G}_7$: In this game, we still deal with the case in which all flows received by the client up to $(S4)$ were oracle-generated. We are now at beginning of $(S4)$ and we want to simulate the (honest) server, without knowing any password, but we now suppose that `flow-three` was not oracle-generated. Note that in this case `flow-one` cannot have been oracle-generated. Otherwise, the signature σ_1 would have been rejected.

Recall that from $\boldsymbol{G}_2$, the simulator extracts the password from the commitment of the client, with the help of the secret key. The extraction can fail with negligible probability if $\mathcal{S}$ recovers two different passwords (thanks to the binding property, see $\boldsymbol{G}_2$ and $\boldsymbol{G}_3$). In such a case, the simulator aborts the game. It can also happen that the extraction issues no password (for example if the values sent in the commitment were not encryptions of 0 or 1). Then, if it has recovered a password, the simulator asks a `GoodPwd` query to the functionality. If it is correct, then it computes honestly the session key of the server. Otherwise, or if it has not recovered any password in the extraction, it sets the values $\mathsf{Hash}(\mathsf{hk}_\mathrm{J}; \rho, (\ell_\mathrm{I}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{I})$ and $\mathsf{ProjHash}(\mathsf{hp}_\mathrm{I}; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{J}; \mathsf{w}_\mathrm{J})$ at random. Recall that if the values sent in the commitment were not encryptions of 0 or 1, the smooth hash value will be random.

The corruptions which can follow this step are dealt with as in the former game. Due to the smoothness of the hash function, this game is indistinguishable from the previous one: The proof is exactly the same as in $\boldsymbol{G}_5$, but it is made more easier, since the key of the client is already random.

Game $\boldsymbol{G}_8$: In this game, we replace `GoodPwd` queries by `TestPwd` ones, and `SamePwd` by `NewKey` ones. If a session aborts or terminates, $\mathcal{S}$ reports it to $\mathcal{A}$. If a session terminates with a session key sk, then $\mathcal{S}$ makes a `NewKey` call to the functionality, specifying the session key sk. But recall that unless the session is compromised, the functionality will ignore the key specified by $\mathcal{S}$.

We show in Appendix F.4 that this game is indistinguishable from the ideal-world experiment.

F Details of the Proof

F.1 Indistinguishability of $\mathbf{G_4}$ and $\mathbf{G_3}$

We now describe what happens in case of corruptions. First, if Alice gets corrupted after the execution of $(U3)$, then the simulator will recover her password and thus be able to compute everything correctly. Recall that the projection key did not depend on the password.

Second, if Bob gets corrupted after $(U3)$ or after $(S4)$, then the simulator will be able to compute everything correctly. In particular in the second case, the adversary will get a coherent value of sk_J. Then, the simulator asks a `GoodPwd` query for Alice with Bob's password to the functionality. If they are the same, $\mathcal{S}$ recovers the password of Alice, and since it hasn't really erased her data, it will be

able to compute sk_I exactly as the adversary should have done it for Bob (in particular because the projection key does not depend on the password). Otherwise, it gives Alice a random key: there is no need that the players get the same key if they don't share the same password – recall that all private data is erased so that the adversary cannot verify anything.

Finally, if Alice gets corrupted by the end of $(U5)$, the simulator will recover her password and ask a `SamePwd` query to the functionality. If σ_J is correct and they share the same password, then the simulator computes sk_I exactly as the adversary should have done it for Bob (one again because the projection key does not depend on the password). Otherwise, if the signature is correct but their passwords are different, then Alice is provided with a random value (recall that her data is supposed to be erased, so that the adversary is not supposed to be able to verify it). Otherwise, if the signature is incorrect, the simulator aborts the game. We can see here the necessity of step $(U5)$ in order to guarantee adaptive security.

We now show that an environment that distinguishes $\boldsymbol{G}_4$ from $\boldsymbol{G}_3$ can be used to construct a distinguisher between `Expt-Hash` and `Expt-Unif` as described in [22] (see Appendix A), violating their Corollary 3.3.

We first define hybrid games H_i as follows. First note that the sessions are ordered with respect to the rounds $(U1)$. In all sessions before the i^{th} one, the computation is random, and in all the sessions afterwards $(i, i+1, \ldots)$, the values are set as real. In all "random" cases, we set $\mathsf{ProjHash}(\mathsf{hp}_I; \rho, (\ell_\mathrm{J}, \mathsf{pw}_\mathrm{J}), \mathsf{com}_\mathrm{J}; \mathsf{w}_J)$ to the same value during the simulation of the server if everything goes well, that is, if no corruption occurred and the passwords are the same. With these notations, the $i = 1$ case is exactly $\boldsymbol{G}_3$ and $i = q_s + 1$ is exactly $\boldsymbol{G}_4$. Pictorially,

$$\begin{cases} \text{Random} & 1, \ldots, i-1 \\ \text{Real} & i, \ldots, q_s \end{cases}$$

Our aim is now to prove the indistinguishability of H_i and H_{i+1}. We define the event E_i, stating that there exists a corruption between the committing $(U1)$ and hashing $(U3)$ steps in the i^{th} session. Notice that the probability of this event is the same in the two following games H_i and H_{i+1}. This is true despite the fact that we consider concurrent sessions. To see this, notice that even though E_i may depend on sessions with a larger index, the only difference between these two games concerns round $(U3)$ of the i^{th} session: everything is identical before this step. Since the corruption occurs before this round, the probability that the adversary corrupts a player in the i^{th} session, which only depends on what happened before, is the same in both cases.

We now denote by $\textsc{out}_\mathcal{Z}$ the output of the environment at the end of the execution and compute the difference between $\Pr[\textsc{out}_\mathcal{Z} = 1]$ in the two different games:

$$\begin{aligned}
&\left|\Pr_{H_{i+1}}[\textsc{out}_\mathcal{Z} = 1] - \Pr_{H_i}[\textsc{out}_\mathcal{Z} = 1]\right| \\
&= \left|\Pr_{H_{i+1}}[\textsc{out}_\mathcal{Z} = 1 \wedge E_i] + \Pr_{H_{i+1}}[\textsc{out}_\mathcal{Z} = 1 \wedge \neg E_i]\right. \\
&\qquad \left. - \Pr_{H_i}[\textsc{out}_\mathcal{Z} = 1 \wedge E_i] - \Pr_{H_i}[\textsc{out}_\mathcal{Z} = 1 \wedge \neg E_i]\right| \\
&\leq \left|\Pr_{H_{i+1}}[\textsc{out}_\mathcal{Z} = 1 \wedge E_i] - \Pr_{H_i}[\textsc{out}_\mathcal{Z} = 1 \wedge E_i]\right| \\
&\qquad + \left|\Pr_{H_{i+1}}[\textsc{out}_\mathcal{Z} = 1 \wedge \neg E_i] - \Pr_{H_i}[\textsc{out}_\mathcal{Z} = 1 \wedge \neg E_i]\right| \\
&\leq \left|\Pr_{H_{i+1}}[\textsc{out}_\mathcal{Z} = 1 | E_i] - \Pr_{H_i}[\textsc{out}_\mathcal{Z} = 1 | E_i]\right| \ \left|\Pr_{H_i}[E_i]\right| \\
&\qquad + \left|\Pr_{H_{i+1}}[\textsc{out}_\mathcal{Z} = 1 | \neg E_i] - \Pr_{H_i}[\textsc{out}_\mathcal{Z} = 1 | \neg E_i]\right| \ \left|\Pr_{H_i}[\neg E_i]\right|
\end{aligned}$$

First consider the first term of this sum. If there is a corruption, we learn the password and also (in fact, we simulate in a coherent way) the randomness, enabling us to compute everything correctly. Thus, this term is equal to zero.

We now consider the second term, corresponding to the case where there is no corruption, and show that $|\Pr[\texttt{Expt-Hash}(D) = 1] - \Pr[\texttt{Expt-Unif}(D) = 1]|$ bounds (to within a negligible amount) the probability that the environment distinguishes H_{i+1} from H_i. More precisely, let's consider the following game H (as described below) in which the oracles `Commit` and `Hash` appear in the i^{th} session only under the assumption $\neg E_i$.

H is as follows: Let D be a machine that receives a randomly chosen public key pk_2 and emulates the game H_i with the following changes for the i^{th} session. On receiving a valid oracle-generated `flow-one`, D does not directly compute c_2 but it queries instead the oracle `Commit`() and sets c_2 to the value returned. If Alice receives the unmodified $\mathsf{com}_{\mathrm{J}}$ in the `flow-two` message, then D queries $\mathtt{Hash}(\mathsf{com}_{\mathrm{J}})$ and receives a pair (s_I, η_I). Then it sets $\mathsf{hp}_I = s_I$ and $\mathsf{ProjHash}(\mathsf{hp}_I; \rho, (\ell_{\mathrm{J}}, \mathsf{pw}_{\mathrm{I}}), \mathsf{com}_{\mathrm{J}}; \mathsf{w}_J)$. Note that w_J was here the randomness used by the oracle `Commit` in the query that generated $\mathsf{com}_{\mathrm{J}}$. Then, if Bob receives the unmodified projected key hp_I, D also uses η_I for the appropriate portion of the session key – in the case they have the same password. Finally, D outputs whatever the environment outputs. It is easy to see that $H = H_{i+1}$ in the case where the oracle `Hash` returns a random value, and it is equal to H_i otherwise. □

F.2 Indistinguishability of $\mathbf{G_5}$ and $\mathbf{G_4}$

First note that the corruptions which can follow this step are dealt with as in the former game, and that this simulation is compatible with the former game.

If the password recovered from the server is correct, the simulation is done honestly, so that this game is perfectly equivalent to the previous one. Otherwise, if the password is incorrect, or if no password was recovered, then $\mathsf{com}_{\mathrm{J}}$ is invalid.

Given an invalid $\mathsf{com}_{\mathrm{J}}$, with $(P_{\mathrm{I}}, \mathrm{ssid})$'s password pw_{I} and the label ℓ_{I}, the distribution $\{\mathsf{pk}_{\mathrm{I}}, \mathsf{com}_{\mathrm{I}}, \ell_{\mathrm{I}}, \mathsf{pw}_{\mathrm{I}}, \mathsf{hp}_{\mathrm{I}}, \mathsf{Hash}(\mathsf{hk}_{\mathrm{I}}; \rho, (\ell_{\mathrm{J}}, \mathsf{pw}_{\mathrm{I}}), \mathsf{com}_{\mathrm{J}})\}$ is statistically close to the distribution $\{\mathsf{pk}_{\mathrm{I}}, \mathsf{com}_{\mathrm{I}}, \ell_{\mathrm{I}}, \mathsf{pw}_{\mathrm{I}}, \mathsf{hp}_{\mathrm{I}}, z\}$ where z is a random element of the group G (due to the smoothness of the hash function). Since $\mathsf{Hash}(\mathsf{hk}_{\mathrm{I}}; \rho, (\ell_{\mathrm{J}}, \mathsf{pw}_{\mathrm{I}}), \mathsf{com}_{\mathrm{J}})$ is a component of the session key sk_{I} for $(P_{\mathrm{I}}, \mathrm{ssid})$, then the session key generated is statistically close to uniform.

Then, since $\mathsf{com}_{\mathrm{J}}$ is invalid, $(P_{\mathrm{J}}, \mathrm{ssid})$ will compute honestly the key, but he will not obtain the same session key since the passwords are different. This behavior is equivalent to what happens in the ideal functionality: the corresponding sessions of $(P_{\mathrm{I}}, \mathrm{ssid})$ and $(P_{\mathrm{J}}, \mathrm{ssid})$ either do not have a matching conversation, or were given different passwords by the environment, so that $(P_{\mathrm{J}}, \mathrm{ssid})$ will not be given the same session key as $(P_{\mathrm{I}}, \mathrm{ssid})$.

F.3 Indistinguishability of $\mathbf{G_6}$ and $\mathbf{G_5}$

The pseudo-randomness of the hash functions shows the indistinguishability between $\boldsymbol{G}_6$ and $\boldsymbol{G}_5$. Indeed, the environment cannot become aware that the keys were chosen at random. More precisely, we do the same manipulation as in the proof of $\boldsymbol{G}_4$, but this time considering the smooth projective hash function Hash with respect to $\mathsf{com}_{\mathrm{J}}$. Note that in the hybrid games, the sessions are ordered with respect to the rounds $(S2)$.

We now consider the case where Bob gets corrupted between $(S4)$ and $(U5)$. Then, the simulator recovers his password and it is able to compute everything correctly (recall that the projection keys do not depend on the passwords). It then asks a `GoodPwd` query for the client. If their passwords are different, Alice is provided with a random key. Otherwise, $\mathcal{S}$ gives her the same key as Bob.

Finally note that for sake of simplicity, we only compute $\mathsf{Hash}_{\mathrm{J}}$ in $(S4)$ in the simulation. But, if the server is corrupted after $(S2)$, the simulator recovers his password and is able to provide the adversary with a correct $\mathsf{Hash}_{\mathrm{J}}$ (once more because the projection key does not depend on the password).

F.4 Indistinguishability between $\mathbf{G_8}$ and the Ideal Game

The only difference between $\boldsymbol{G}_7$ and $\boldsymbol{G}_8$ is that the `GoodPwd` queries are replaced by `TestPwd` queries to the functionality and the `SamePwd` by a `NewKey` query. We say that the players have matching sessions if they share the same *ssid* and if they agree on the values of $\mathsf{com}_{\mathrm{I}}$, $\mathsf{com}_{\mathrm{J}}$, hp_I and hp_J (that is, all the values that determine the key). We now show that $\boldsymbol{G}_8$ and IWE are indistinguishable.

First, if both players share the same password and remain honest until the end of the game, and if there are no impersonations, they will obtain a random key, both in $\boldsymbol{G}_8$ (from $\boldsymbol{G}_6$) and IWE, as

there are no `TestPwd` queries and the sessions remain fresh. Second, if they share the same password but there are impersonation attempts, then they receive independently-chosen random keys (from $\boldsymbol{G}_5$ or $\boldsymbol{G}_7$). Third, if they don't share the same password, then they get independently-chosen random keys. Now, we need to show that two players will receive the same key in $\boldsymbol{G}_8$ if and only if it happens in IWE.

First consider the case of players with matching session and the same password. If both players remain honest until the end of the game, they will receive the same key from $\boldsymbol{G}_6$. If not, it will still be the same from $\boldsymbol{G}_5$ or $\boldsymbol{G}_7$. Recall that if there are some impersonation attempts, the keys will be random and independent, both in $\boldsymbol{G}_8$ and IWE. In IWE, the functionality will receive two `NewSession` queries with the same password. If both are honest, it will not receive any `TestPwd` query, so that the key will be the same for the two players. And if one is corrupted and a `TestPwd` query is done (and correct, since they have the same password), then they will also have the same key, chosen by the adversary.

Then, consider the case of players with matching session but not the same password. If both players remain honest until the end of the game, they will receive independently-chosen random keys from $\boldsymbol{G}_6$. If not, it will still be the same from $\boldsymbol{G}_7$. In IWE, the functionality will receive two `NewSession` queries with different passwords. It will give them different keys by definition.

Finally, consider the case of players with no matching session. It is clear that in $\boldsymbol{G}_8$ the session keys of those players will be independent because they are not set in any of the games. In IWE, the only way that they receive matching keys is that the functionality receives two `NewSession` queries with the same password, and $\mathcal{S}$ sends a `NewKey` query for these sessions without having sent any `TestPwd` queries. But if the two sessions do not have a matching conversation, they must differ in either $\mathsf{com}_{\mathrm{I}}$, $\mathsf{com}_{\mathrm{J}}$, hp_I or hp_J. In this case, they will refuse the signature of the other player and abort the game.

If one of the player is corrupted by the end of the game, the simulator recovers its password and uses it in a `TestPwd` query for the other player to the functionality, as explained in $\boldsymbol{G}_4$. If the result is correct, then both players are give the same key. Otherwise, they are given independently-chosen random keys. This is exactly the behavior of the functionality in IWE.