New Smooth Projective Hash Functions and One-Round Authenticated Key Exchange

Fabrice Ben Hamouda[1], Olivier Blazy[2], Céline Chevalier[3], David Pointcheval[1], and Damien Vergnaud[1]

[1] ENS, Paris, France †
[2] Ruhr-Universität Bochum, Germany
[3] Université Paris II, France

Abstract. *Password-Authenticated Key Exchange* (PAKE) has received deep attention in the last few years, with a recent improvement by Katz-Vaikuntanathan, and their one-round protocols: the two players just have to send simultaneous flows to each other, that depend on their own passwords only, to agree on a shared high entropy secret key. To this aim, they followed the Gennaro-Lindell approach, with a new kind of *Smooth-Projective Hash Functions* (SPHF). They came up with the first concrete one-round PAKE, secure in the Bellare-Pointcheval-Rogaway model, but at the cost of a simulation-sound NIZK, which makes the overall construction not really efficient.
This paper follows their path with a new efficient instantiation of SPHF on Cramer-Shoup ciphertexts. It then leads to the design of the most efficient PAKE known so far: a one-round PAKE with two simultaneous flows consisting of 6 group elements each only, in any DDH-group without any pairing. We thereafter show a generic construction for SPHFs, in order to check the validity of complex relations on encrypted values. This allows to extend this work on PAKE to the more general family of protocols, termed *Langage-Authenticated Key Exchange* (LAKE) by Ben Hamouda-Blazy-Chevalier-Pointcheval-Vergnaud, but also to blind signatures.

Keywords. Authenticated Key Exchange, Blind Signatures, Smooth Projective Hash Functions

1 Introduction

Authenticated Key Exchange protocols are quite important primitives for practical applications, since they enable two parties to generate a shared high entropy secret key, to be later used with symmetric primitives to protect communications, while interacting over an insecure network under the control of the adversary. Various authentication means have been proposed, and the most practical one is definitely a shared low entropy secret, or a password they can agree on over the phone, hence PAKE, for *Password-Authenticated Key Exchange*. The most famous instantiation has been proposed by Bellovin and Merritt [BM92], EKE for Encrypted Key Exchange, which simply consists of a Diffie-Hellman key exchange [DH76], where the flows are symmetrically encrypted under the shared password. Overall, the equivalent of 2 group elements have to be sent.

A first formal security model was proposed by Bellare, Pointcheval and Rogaway [BPR00] (BPR), to model resistance against off-line dictionary attacks, which essentially says that the best attack is the on-line exhaustive search with all the possible passwords against the server. Several variants of EKE and BPR-security proofs have been proposed in the ideal-cipher model or the random-oracle model [Poi12]. Katz, Ostrovsky and Yung [KOY01] proposed the first practical scheme (KOY), provably secure in the standard model under the DDH assumption. This was a 3-flow protocol, with the client sending 5 group elements plus a verification key and a signature, for a one-time signature scheme, and the server sending 5 group elements. It has been generalized by Gennaro and Lindell [GL03], making use of smooth projective hash functions.

Smooth Projective Hash Functions (SPHFs) have been introduced by Cramer and Shoup [CS02] in order to build IND-CCA encryption schemes [CS98]. They can be seen as a kind of implicit designated-verifier proofs of membership [ACP09, BPV12]. Basically, SPHFs are families of pairs of functions (Hash, ProjHash) defined on a language L. These functions are indexed by a pair of associated keys (hk, hp), where hk, the hashing key, can be seen as a private key and hp, the projection key, as the public key. On a word $W \in$ L, both

† CNRS – UMR 8548 and INRIA – EPI Cascade

functions should lead to the same result: $\mathsf{Hash}(\mathsf{hk}, \mathrm{L}, W)$ with the hashing key and $\mathsf{ProjHash}(\mathsf{hp}, \mathrm{L}, W, w)$ with the projection key only but also a witness w that $W \in \mathrm{L}$. Of course, if $W \notin \mathrm{L}$, such a witness does not exist, and the smoothness property states that $\mathsf{Hash}(\mathsf{hk}, \mathrm{L}, W)$ is independent of hp, and thus even knowing hp, one cannot guess $\mathsf{Hash}(\mathsf{hk}, \mathrm{L}, W)$. The Gennaro and Lindell approach has thereafter been applied to the *Universal Composability* (UC) framework by Canetti *et al.* [CHK$^+$05], for static corruptions only, and improved by Abdalla, Chevalier and Pointcheval [ACP09] to resist to adaptive adversaries. But the 3-flow KOY protocol remains the most efficient protocol BPR-secure under the DDH assumption.

One-Round PAKE. More recently, the ultimate step for PAKE has been achieved by Katz and Vaikuntanathan [KV11] (KV), who proposed a *practical* one-round PAKE, where the two players just have to send simultaneous flows to each other, that depend on their own password only. More precisely, each flow just consists of an IND-CCA ciphertext of the password and an SPHF projection key for the correctness of the partner's ciphertext (the word is the ciphertext and the witness is the random coins of the encryption). The shared secret key is eventually the product of the two hash values. Because of the simultaneous flows, one flow cannot depend on the partner's flow, which makes impossible the use of the Gennaro and Lindell SPHF (later named GL-SPHF), in which the projection key depends on the word (the ciphertext here). On the other hand, the adversary can wait for the player to send his flow first, and then adapt its message, which requires stronger security notions than the initial Cramer-Shoup SPHF (lated named CS-SPHF), in which the smoothness does not hold anymore if the word is generated after having seen the projection key. This led Katz and Vaikuntanathan to provide a new definition for SPHF (later named KV-SPHF), where the projection key depends on the hashing key only, and the smoothness holds even if the word is chosen after having seen the projection key. Variations between CS-SPHF, GL-SPHF and KV-SPHF indeed rely in the way one computes hp from hk and W, but also on the smoothness property, according to the freedom the adversary has to choose W.

The strongest definition is the recent KV-SPHF, however previous SPHFs known for Cramer-Shoup ciphertexts were GL-SPHF only, because the projection key hp had to be computed for a specific ciphertext. For their PAKE, Katz and Vaikuntanathan did not manage to construct such a KV-SPHF on efficient IND-CCA encryption schemes. They then suggested to use the Naor and Yung approach [NY90], with any IND-CPA encryption scheme and a simulation-sound NIZK (SS-NIZK) [Sah99]. Such an SS-NIZK is quite costly, and they also suggested to use Groth-Sahai [GS08] proofs in bilinear groups and thus with the linear encryption [BBS04]. The instantiation proven under the DLin assumption then uses a ciphertext consisting of 66 group elements and a projection key consisting of 4 group elements. As a consequence, the two players have to send 70 group elements each, which is far more costly than the KOY protocol, but it is one-round only.

More recent results on SS-NIZKs or IND-CCA encryption schemes improved on that: Libert and Yung [LY12] proposed a more efficient SS-NIZK of plaintext equality in the Naor-Yung-type cryptosystem. The proof can be reduced from 60 to 22 group elements and the communication complexity of the resulting PAKE is decreased to 32 group elements per user. Jutla and Roy [JR12] proposed relative-sound NIZKs as an efficient alternative to SS-NIZKs to build new publicly-verifiable IND-CCA encryption schemes. They can then decrease the PAKE communication complexity to 20 group elements per user. Note they all need pairing computations.

Achievements. We first revisit the different definitions for SPHFs proposed in [CS02, GL03, KV11], denoted respectively CS-SPHF, GL-SPHF and KV-SPHF. While CS-SPHF was the initial definition useful for IND-CCA encryption, GL-SPHF and KV-SPHF did prove quite useful too.

We propose a generic framework for constructing such SPHFs that aim at proving that a tuple of ElGamal-like ciphertexts contains plaintexts that satisfy a system of equations. It is inspired from [GGH12] to construct SPHFs on graded rings, which encompass both classical and bilinear groups. More concretely, we provide an instantiation of KV-SPHF on a Cramer-Shoup ciphertext, which can then be used within the above KV framework for one-round PAKE [KV11], in the BPR security model. Our scheme just consists of 6 group elements in each direction under the DDH assumption (4 for the ciphertext, and 2 for the projection key). This has to be compared with the 20 group elements in the best improvements discussed above, which all need pairing-friendly groups and pairing computations, or with the KOY protocol that has the same complexity, but with three sequential flows. Thanks to the general systems that can be handled by our generic framework, we apply our $\mathsf{KV\text{-}SPHF}$ to the *Language-Authenticated Key-Exchange* (LAKE) protocols introduced in [BBC+12], and provide the first one-round LAKE protocols. We also detail the first SPHFs able to handle multi-exponentiation equations, without requiring pairings. They are thus quite efficient.

We also improve on the *blind signature* scheme presented in [BPV12], from $5\ell + 6$ group elements in $\mathbb{G}_1$ and 1 group element in $\mathbb{G}_2$ to $3\ell + 7$ group elements in $\mathbb{G}_1$ and 1 group element in $\mathbb{G}_2$, for an ℓ-bit message to be blindly signed with a Waters signature [Wat05].

Language Definition. We could cover languages as general as those covered in [BBC+12], but for the sake of clarity, and since the main applications need some particular cases only, we focus on languages of ciphertexts, which can be seen as perfectly binding commitments possibly extractable, denoted $\textsc{LofC}_{\mathsf{full\text{-}aux}}$. The parameter $\mathsf{full\text{-}aux}$ will parse in two parts $(\mathsf{crs}, \mathsf{aux})$: the public part crs, known in advance, and the private part aux, possibly chosen later. More concretely, crs represents the public values: it will define the encryption scheme (and will thus contain the global parameters and the public key of the encryption scheme) with the global format of both the tuple to be encrypted and the equation it should satisfy, and possibly additional public coefficients; while aux represents the private values: it will specify this equation, with more coefficients or constants that will remain private. We will consider equations that are efficiently verifiable on the plaintexts, which means that the witnesses w, enough to check whether a tuple of ciphertexts is in $\textsc{LofC}_{\mathsf{full\text{-}aux}}$ for those who know $\mathsf{full\text{-}aux}$, are the plaintext M and the random coins r used for encryption.

A generalization of AKE protocols has been recently proposed, so-called *Language-Authenticated Key Exchange* (LAKE) [BBC+12]: it allows two users, Alice and Bob, each owning a word in a specific language, to agree on a shared high entropy secret if each user knows a word in the language the other thinks about. More precisely, they first both agree on public parameters pub, but Bob will think about *priv* for his expected Alice's value of priv, Alice will do the same with $priv'$ for Bob's private value priv'; eventually, if $priv = \mathsf{priv}$ and $priv' = \mathsf{priv}'$, and if they both know words in the appropriate languages, then the key agreement will succeed. In case of failure, no information should leak to the players about the reason of failure, except that the inputs did not satisfy the relations, or the languages were not consistent. Eavesdroppers do not even learn the outcome.

Their construction follows the Gennaro and Lindell approach for PAKE [GL03]: each player commits to the private values (their own value priv, and the expected partner's value $priv'$) as well as their own word, and projection keys are sent to compute random values that will be the same if and only if everything is consistent. To achieve one-round LAKE, one also needs $\mathsf{KV\text{-}SPHF}$ on ciphertexts for plaintext-equality tests (equality of the private values and expected private values) and for language-membership. But some parameters of the languages have to remain hidden, hence the aux part that should not leak from hp, and we will thus use SPHFs for which hp depends on crs but not on aux.

Outline of the Paper

In Section 2, we first review SPHFs, with the various cases that already appeared in the literature. We then illustrate them on Cramer-Shoup ciphertexts, giving the first known $\mathsf{KV\text{-}SPHF}$ construction. We can immediately apply it to the KV framework for PAKE [KV11], and obtain a highly-efficient one-round PAKE. We also show how it extends to LAKE, and formally prove the security of our generic LAKE construction in Section 3. This extension gives concrete one-round LAKE instantiations as soon as one has concrete $\mathsf{KV\text{-}SPHF}$ constructions for languages of ciphertexts. Therefore, in Section 4, we provide a general framework to build SPHFs on ElGamal-like ciphertexts, with some concrete illustrations in Section 5. We eventually provide another application to blind signatures in Section 6. Because of the lack of space, several details or proofs will be postponed to the Appendix.

2 New SPHFs on Cramer-Shoup Ciphertexts

In this section, we first recall the definitions of SPHFs and present our classification based on the dependence between words and keys. According to this classification, there are three types of SPHFs: the initial Cramer-Shoup [CS02] type (CS-SPHF) introduced for IND-CCA encryption, the Gennaro-Lindell [GL03] type (GL-SPHF) introduced for PAKE, and the Katz-Vaikuntanathan [KV11] type (KV-SPHF) introduced for one-round PAKE.

After a quick review on the Cramer-Shoup encryption scheme, we introduce our new KV-SPHF for Cramer-Shoup ciphertexts which immediately leads to a quite efficient instantiation of the KV one-round PAKE [KV11]. We thereafter show how KV-SPHF can be used to efficiently implement a one-round LAKE [BBC+12], with a complete security proof that also proves our one-round PAKE.

2.1 General Definition of SPHFs

Let us consider a language $\mathrm{L} \subseteq \mathcal{S}et$, and some global parameters for the SPHF, assumed to be in the common random string (CRS). The SPHF system for the language L is defined by four algorithms:

- $\mathsf{HashKG}(\mathrm{L})$ generates a hashing key hk for the language L;
- $\mathsf{ProjKG}(\mathsf{hk}, \mathrm{L}, C)$ derives the projection key hp, possibly depending on the word C;
- $\mathsf{Hash}(\mathsf{hk}, \mathrm{L}, C)$ outputs the hash value from the hashing key;
- $\mathsf{ProjHash}(\mathsf{hp}, \mathrm{L}, C, w)$ outputs the hash value from the projection key hp, and the witness w.

The *correctness* of the SPHF assures that if $C \in \mathrm{L}$ with w a witness of this membership, then the two ways to compute the hash values give the same result: $\mathsf{Hash}(\mathsf{hk}, \mathrm{L}, C) = \mathsf{ProjHash}(\mathsf{hp}, \mathrm{L}, C, w)$. On the other hand, the security is defined through the *smoothness*, which guarantees that, if $C \notin \mathrm{L}$, the hash value is *statistically* indistinguishable from a random element, even knowing hp. For that, we use the classical notion of statistical distance recalled in the Appendix A.2.

2.2 Smoothness Adaptivity and Key Word-Dependence

This paper will exploit the very strong notion KV-SPHF. Informally, while the GL-SPHF definition allowed the projection key hp to depend on the word C, the KV-SPHF definition prevents the projection key hp to depend on C, as in the original CS-SPHF definition. In addition, the smoothness should hold even if C is chosen as an arbitrary function of hp. This models the fact the adversary can see hp before deciding which word C he is interested in. More formal definitions follow, where we denote Π the range of the hash function.

CS-SPHF. This is the initial definition of SPHF, where the projection key hp does not depend on the word C (word-independent key), and the word C cannot be chosen from hp either for breaking the smoothness (non-adaptive smoothness). More formally, a CS-SPHF is ε-smooth if ProjKG does not use its input C and if, for any $C \in \mathcal{S}et \backslash \mathrm{L}$, the two following distributions are ε-close:

$$\{(\mathsf{hp}, H) \mid \mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathrm{L});\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathrm{L}, \perp);\ H \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathrm{L}, C)\}$$
$$\{(\mathsf{hp}, H) \mid \mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathrm{L});\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathrm{L}, \perp);\ H \stackrel{\$}{\leftarrow} \Pi\}.$$

GL-SPHF. This is a relaxation, where the projection key hp can depend on the word C (word-dependent key). More formally, a GL-SPHF is ε-smooth if, for any $C \in \mathcal{S}et \backslash \mathrm{L}$, the two following distributions are ε-close:

$$\{(\mathsf{hp}, H) \mid \mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathrm{L});\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathrm{L}, C);\ H \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathrm{L}, C)\}$$
$$\{(\mathsf{hp}, H) \mid \mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathrm{L});\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathrm{L}, C);\ H \stackrel{\$}{\leftarrow} \Pi\}.$$

KV-SPHF. This is the strongest SPHF, the projection key hp does not depend on the word C (word-independent key) and the smoothness holds even if C depends on hp (adaptive smoothness). More formally, a KV-SPHF is ε-smooth if ProjKG does not use its input C and, for any function f onto $\mathcal{S}et\backslash\mathrm{L}$, the two following distributions are ε-close:

$$\{(\mathsf{hp}, H) \mid \mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathrm{L});\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathrm{L}, \perp);\ C \leftarrow f(\mathsf{hp});\ H \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathrm{L}, C)\}$$
$$\{(\mathsf{hp}, H) \mid \mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathrm{L});\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathrm{L}, \perp);\ C \leftarrow f(\mathsf{hp});\ H \stackrel{\$}{\leftarrow} \mathit{\Pi}\}.$$

Remark 1. One can see that a perfectly smooth (*i.e.*, 0-smooth) CS-SPHF is also a perfectly smooth KV-SPHF, since each value H has exactly the same probability to appear, and so choosing C adaptively does not increase the distance. However, as soon as a weak word C can bias the distribution, f can exploit it.

2.3 SPHF on Languages of Ciphertexts

In the following, we focus on languages of ciphertexts, where the witnesses are the random coins used for encryption and possibly the plaintexts. The languages $\mathrm{LOFC}_{\mathsf{full\text{-}aux}}$ will thus be defined by $\mathsf{full\text{-}aux} = (\mathsf{crs}, \mathsf{aux})$, where crs will correspond to some public parameters and aux to some parameters to be kept secret. The first part crs will at least contain the global parameters and the public key of the encryption, but possibly (public) additional parameters on the relation the plaintexts should satisfy.

To keep aux secret, hp should not leak any information. We will thus restrict HashKG and ProjKG not to use the parameter aux, but just crs. This is a stronger restriction than required for our purpose, since one can use aux without leaking any information about it. But we already have quite efficient instantiations, and it makes everything much simpler to present.

2.4 SPHFs on Cramer-Shoup Ciphertexts

Labeled Cramer-Shoup Encryption Scheme (CS). The CS labeled encryption scheme is recalled in the Appendix A.3. We briefly review it here by combining all the public information in the encryption key. We thus have a group $\mathbb{G}$ of prime order p, with two independent generators $(g_1, g_2) \stackrel{\$}{\leftarrow} \mathbb{G}^2$, a hash function $\mathfrak{H}_K \stackrel{\$}{\leftarrow} \mathcal{H}$ from a collision-resistant hash function family onto $\mathbb{Z}_p^*$, and a reversible mapping $\mathcal{G}$ from $\{0,1\}^n$ to $\mathbb{G}$. From 5 scalars $(x_1, x_2, y_1, y_2, z) \stackrel{\$}{\leftarrow} {\mathbb{Z}_p}^5$, one also sets $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$. The encryption key is $\mathsf{ek} = (\mathbb{G}, g_1, g_2, c, d, h, \mathfrak{H}_K)$, while the decryption key is $\mathsf{dk} = (x_1, x_2, y_1, y_2, z)$. For the message $m \in \{0,1\}^n$, with $M = \mathcal{G}(m) \in \mathbb{G}$ and $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e) \in \mathbb{Z}_p^*$, the labeled Cramer-Shoup is:

$$C \stackrel{\mathsf{def}}{=} \mathsf{CS}(\ell, \mathsf{ek}, M; r) \stackrel{\mathsf{def}}{=} (\mathbf{u} = (g_1^r, g_2^r), e = M \cdot h^r, v = (cd^\xi)^r).$$

If one wants to encrypt a vector of messages, encoded as group elements $(M_1, \ldots, M_n)$, all at once in a non-malleable way, one computes all the individual ciphertexts with a common $\xi = \mathfrak{H}_K(\ell, \mathbf{u}_1, \ldots, \mathbf{u}_n, e_1, \ldots, e_n)$ for the v's. Hence, everything done on tuples of ciphertexts will work on ciphertexts of vectors.

The (known) GL-SPHF for CS. Gennaro and Lindell [GL03] proposed an SPHF on labeled Cramer-Shoup ciphertexts: the hashing key just consists of a random tuple $\mathsf{hk} = (\eta_1, \eta_2, \lambda, \mu) \stackrel{\$}{\leftarrow} \mathbb{Z}_p^4$. The associated projection key, on a ciphertext $C = (\mathbf{u} = (g_1^r, g_2^r), e = \mathcal{G}(m) \cdot h^r, v = (cd^\xi)^r)$, is $\mathsf{hp} = g_1^{\eta_1} g_2^{\eta_2} h^\lambda (cd^\xi)^\mu \in \mathbb{G}$. Then, one can compute the hash value in two different ways, for the language $\mathrm{LOFC}_{\mathsf{ek},m}$ of the valid ciphertexts of $M = \mathcal{G}(m)$:

$$H \stackrel{\mathsf{def}}{=} \mathsf{Hash}(\mathsf{hk}, (\mathsf{ek}, m), C) \stackrel{\mathsf{def}}{=} u_1^{\eta_1} u_2^{\eta_2} (e/\mathcal{G}(m))^\lambda v^\mu = \mathsf{hp}^r \stackrel{\mathsf{def}}{=} \mathsf{ProjHash}(\mathsf{hp}, (\mathsf{ek}, m), C, r) \stackrel{\mathsf{def}}{=} H'.$$

– Players U and U' both use $\mathsf{ek} = (\mathbb{G}, g_1, g_2, c, d, h, \mathfrak{H}_K)$
– U, with password pw, chooses $\mathsf{hk} = (\eta_1, \eta_2, \theta, \lambda, \mu) \stackrel{\$}{\leftarrow} \mathbb{Z}_p^5$, and computes $\mathsf{hp} = (\mathsf{hp}_1 = g_1^{\eta_1} g_2^{\theta} h^{\lambda} c^{\mu}, \mathsf{hp}_2 = g_1^{\eta_2} d^{\mu})$.
U sets $\ell = (U, U', \mathsf{hp})$ and generates $C = (\mathbf{u} = (g_1^r, g_2^r), e = \mathcal{G}(\mathsf{pw}) \cdot h^r, v = (cd^{\xi})^r)$ where $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$.
It sends $\mathsf{hp} \in \mathbb{G}^2$ and $C \in \mathbb{G}^4$ to U'.
– Upon receiving $\mathsf{hp}' = (\mathsf{hp}'_1, \mathsf{hp}'_2) \in \mathbb{G}^2$ and $C' = ((u'_1, u'_2), e', v') \in \mathbb{G}^4$ from U', it sets $\ell' = (U', U, \mathsf{hp}')$ and $\xi' = \mathfrak{H}_K(\ell', \mathbf{u}', e')$ and computes $\mathsf{sk}_U = {u'_1}^{(\eta_1 + \xi' \eta_2)} {u'_2}^{\theta} (e'/\mathcal{G}(\mathsf{pw}))^{\lambda} {v'}^{\mu} \times (\mathsf{hp}'_1 {\mathsf{hp}'_2}^{\xi})^r$

Fig. 1. One-Round PAKE based on DDH

A (new) KV-SPHF for CS. We give here the description of the first known KV-SPHF for labeled Cramer-Shoup ciphertexts, whose proof can be found in Section 4 as an illustration of our new formalism: the hashing key just consists of a random tuple $\mathsf{hk} = (\eta_1, \eta_2, \theta, \lambda, \mu) \stackrel{\$}{\leftarrow} \mathbb{Z}_p^5$. The associated projection key is the pair $\mathsf{hp} = (\mathsf{hp}_1 = g_1^{\eta_1} g_2^{\theta} h^{\lambda} c^{\mu}, \mathsf{hp}_2 = g_1^{\eta_2} d^{\mu}) \in \mathbb{G}^2$. Then one can compute the hash value in two different ways, for the language $\mathrm{LofC}_{\mathsf{ek},m}$ of the valid ciphertexts of $M = \mathcal{G}(m)$:

$$H = \mathsf{Hash}(\mathsf{hk}, (\mathsf{ek}, m), C) \stackrel{\text{def}}{=} u_1^{(\eta_1 + \xi\eta_2)} u_2^{\theta} (e/\mathcal{G}(m))^{\lambda} v^{\mu} = (\mathsf{hp}_1 \mathsf{hp}_2^{\xi})^r \stackrel{\text{def}}{=} \mathsf{ProjHash}(\mathsf{hp}, (\mathsf{ek}, m), C, r) = H'.$$

Theorem 2. *The above* SPHF *is a* 0*-smooth (or perfectly smooth)* KV-SPHF*.*

3 Application: LAKE and Efficient One-Round PAKE

3.1 An Efficient One-Round PAKE

Review of Katz-Vaikuntanathan's PAKE. As explained earlier, Katz and Vaikuntanathan [KV11] recently proposed a one-round PAKE. Their general framework follows Gennaro and Lindell [GL03] approach: each player sends an encryption of the password, and then uses an SPHF on the partner's ciphertext to check whether it actually contains the same password. The two hash values are multiplied to produce the session key. If the encrypted passwords are the same, the different ways to compute the hash values give the same results. If the passwords differ, the smoothness makes them independent. To this aim, the authors need an SPHF on a labeled IND-CCA encryption scheme. To allow a one-round PAKE, the ciphertext and the projection key on the partner's ciphertext should be sent together, before having seen the partner's ciphertext: the projection key should be independent of the ciphertext. On the other hand, the adversary can wait to have received the partner's projection key before generating the ciphertext, and thus a stronger smoothness is required. This is exactly what we called a KV-SPHF in the previous section.

Our Construction. Our KV-SPHF on CS presented in the previous section can be applied to the KV approach for PAKE [KV11]. It leads to the most efficient PAKE known so far, and it is *one-round*. Each user indeed only sends 6 elements of $\mathbb{G}$ (see Figure 1), instead of 70 elements of $\mathbb{G}$ for the KV instantiation using a Groth-Sahai SS-NIZK [GS08], or 20 group elements for the Jutla-Roy [JR12] improvement using a relatively-sound NIZK.

The formal security result follows from the Theorem 4 below. We want to insist that our construction does not need pairing-friendly groups, and the plain DDH assumption is enough, whereas the recent constructions made heavy use of pairing-based proofs. Under the DLin assumption (which is a weaker assumption in any group), still without requiring pairing-friendly groups, our construction would make each user to send 9 group elements only.

3.2 One-Round LAKE

Review of Language-Authenticated Key Exchange. LAKE is a general framework [BBC+12] that generalizes AKE primitives: each player U owns a word W in a certain language L and expects the other player to own a word W' in a language L$'$. If everything is compatible (*i.e.*, the languages are the expected languages and the words are indeed in the appropriate languages), the players compute a common high-entropy secret key, otherwise they learn nothing about the partner's values. In any case, external eavesdroppers do not learn anything, even not the outcome of the protocol: did it succeed or not?

More precisely, we assume the two players have initially agreed on a common public part pub for the languages, but then they each encrypt the private part priv for the language L they want to use, the private part $priv'$ for the language L' they assume the other player will use, and the word W they own in their language L. We will thus have to use SPHFs on these ciphertexts, with $\mathsf{crs} = (\mathsf{ek}, \mathsf{pub})$ and $\mathsf{aux} = \mathsf{priv}$. For simple languages, this encompasses PAKE and Verifier-based PAKE. We refer to [BBC+12] for more applications of LAKE.

A New Security Model for LAKE. The first security model for LAKE [BBC+12] has been given in the UC framework [Can01], as an extension of the UC security for PAKE [CHK+05]. In this paper, we propose an extension of the PAKE security model presented by Bellare, Pointcheval, and Rogaway [BPR00] model for LAKE: the adversary $\mathcal{A}$ plays a find-then-guess game against n players $(P_i)_{i=1,\dots,n}$. It has access to several instances Π_U^s for each player $U \in \{P_i\}$ and can activate them (in order to model concurrent executions) via several queries: Execute-queries model passive eavesdroppings; Send-queries model active attacks; Reveal-queries model a possible bad later use of the session key; the Test-query models the secrecy of the session key. The latter query that has to be asked to a *fresh* instance (which basically means that the session key is not trivially known to the adversary) models the fact that the session key should look random for an outsider adversary. More details can be found in [BPR00].

The quality of an adversary is measured by its ability to distinguish the answer of the Test-query on a fresh instance: a trivial attack is the so-called on-line dictionary attack which consists in trying all the possibilities when interacting with a target player. For PAKE schemes, the advantage of such an attack is q_s/N, where q_s is the number of Send-queries and N the number of possible passwords. A secure PAKE scheme should guarantee this is the best attack, or equivalently that the advantage of any adversary is bounded by $q_s \times 2^{-m}$, where m is the min-entropy of the password distribution. For LAKE, the trivial attack consists in trying all the possibilities for $\mathsf{priv}, priv'$ with a word W in $\mathrm{L}_{\mathsf{pub},\mathsf{priv}}$.

Definition 3 (Security for LAKE). *A LAKE scheme is claimed secure if the advantage of any adversary running in time t is bounded by $q_s \times 2^{-m} \times \mathsf{Succ}^{\mathrm{L}}(t) + \mathsf{negl}()$, where m is the min-entropy of the pair $(\mathsf{priv}, \mathsf{priv}')$, and $\mathsf{Succ}^{\mathrm{L}}(t)$ is the maximal success an adversary can get in finding a word in any $\mathrm{L}_{\mathsf{pub},\mathsf{priv}}$ within time t.*

Our Instantiation. Using the same approach as KV for one-round PAKE [KV11], one can design the scheme proposed on Figure 2, in which both users U and U' use the encryption key ek and the public part pub. This defines $\mathsf{crs} = (\mathsf{ek}, \mathsf{pub})$. When running the protocol, U owns a word W for a private part priv, and thinks to a private part $priv'$ for U', while U' owns a word W' for a private part priv', and thinks to a private $priv$ for U.

This gives a concrete instantiation of one-round LAKE as soon as one can design a KV-SPHF on the language $\mathrm{LoFC}_{(\mathsf{ek},\mathsf{pub}),\mathsf{priv}}$. More precisely, for each player who encrypts $(\mathsf{priv}, priv', W)$, we combine three SPHFs: two on equality-test for the plaintexts priv and $priv'$, and one on $\mathrm{LoFC}_{(\mathsf{ek},\mathsf{pub}),\mathsf{priv}}$ for the ciphertext of $W \in \mathrm{L}_{\mathsf{pub},\mathsf{priv}}$. We stress that hk and hp can depend on crs but not on aux, hence the notations used in the Figure 2. In order to achieve this aim, we give in Section 4 a generic framework to construct such KV-SPHF, and concrete constructions in Section 5. Using a similar proof as in [KV11], one can state (more details on the security model and the full proof can be found in the Appendix B):

Theorem 4. *If the encryption scheme is IND-CCA, and $\mathrm{LoFC}_{(\mathsf{ek},\mathsf{pub}),\mathsf{priv}}$ languages admit KV-SPHFs, then our LAKE protocol is secure.*

From LAKE to PAKE. One can remark that this theorem immediately proves the security of our PAKE from Figure 1: one uses $\mathsf{priv} = \mathsf{priv}' = \mathsf{pw}$ and $\mathsf{pub} = \emptyset$, for the language of the ciphertexts of pw.

- Players U and U' both use ek and agreed on pub.
- U, with $(\mathsf{priv}, \mathit{priv}', W)$, generates $\mathsf{hk} = \mathsf{HashKG}(\mathsf{ek}, \mathsf{pub})$ and $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathsf{ek}, \mathsf{pub}), \perp)$.
  U computes $\ell = (U, U', \mathsf{hp})$ and $C = \mathsf{Encrypt}(\ell, \mathsf{ek}, (\mathsf{priv}, \mathit{priv}', W); r)$, and sends hp, C to U'.
- Upon receiving hp', C' from U', it sets $\ell' = (U', U, \mathsf{hp}')$.
  Eventually, U computes $\mathsf{sk} = \mathsf{Hash}(\mathsf{hk}, ((\mathsf{ek}, \mathsf{pub}), \mathit{priv}'), C') \times \mathsf{ProjHash}(\mathsf{hp}', ((\mathsf{ek}, \mathsf{pub}), \mathsf{priv}), C, r)$.

Fig. 2. One-Round LAKE

4 Generic Framework for SPHFs

In this section, we propose a generic framework for SPHFs using a new notion of graded rings, derived from [GGH12]. It enables to deal with cyclic groups, bilinear groups (with symmetric or asymmetric pairings), or even groups with multi-linear maps. Namely, it handles all the previous constructions from [BBC+12].

4.1 Graded Rings

Our graded rings are a practical way to manipulate elements of various groups involved with pairings, and more generally, with multi-linear maps. This is a slight variant of the notion of graded encoding proposed in [GGH12], where each element has only one representation, instead of a set of representations, and where we can add two elements even with different indexes.

Indexes Set. As in [GGH12], let us consider a finite set of indexes $\Lambda = \{0, \ldots, \kappa\}^\tau \subset \mathbb{N}^\tau$. In addition to considering the addition law $+$ over Λ, we also consider Λ as a bounded lattice, with the two following laws:

$$\sup(\boldsymbol{v}, \boldsymbol{v}') = (\max(\boldsymbol{v}_1, \boldsymbol{v}'_1), \ldots, \max(\boldsymbol{v}_\tau, \boldsymbol{v}'_\tau)) \qquad \inf(\boldsymbol{v}, \boldsymbol{v}') = (\min(\boldsymbol{v}_1, \boldsymbol{v}'_1), \ldots, \min(\boldsymbol{v}_\tau, \boldsymbol{v}'_\tau)).$$

We also write $\boldsymbol{v} < \boldsymbol{v}'$ (resp. $\boldsymbol{v} \leq \boldsymbol{v}'$) if and only if for all $i \in \{1, \ldots, \tau\}$, $\boldsymbol{v}_i < \boldsymbol{v}'_i$ (resp. $\boldsymbol{v}_i \leq \boldsymbol{v}'_i$). Let $\bar{0} = (0, \ldots, 0)$ and $\top = (\kappa, \ldots, \kappa)$, be the minimal and maximal elements.

Graded Ring. The (κ, τ)-graded ring for a commutative ring R is the set $\mathfrak{G} = \Lambda \times R = \{[\boldsymbol{v}, x] \mid \boldsymbol{v} \in \Lambda, x \in R\}$, where $\Lambda = \{0, \ldots, \kappa\}^\tau$, with two binary operations $(+, \cdot)$ defined as follows:

- for every $u_1 = [\boldsymbol{v}_1, x_1], u_2 = [\boldsymbol{v}_2, x_2] \in \mathfrak{G}$: $u_1 + u_2 \stackrel{\text{def}}{=} [\sup(\boldsymbol{v}_1, \boldsymbol{v}_2), x_1 + x_2]$;
- for every $u_1 = [\boldsymbol{v}_1, x_1], u_2 = [\boldsymbol{v}_2, x_2] \in \mathfrak{G}$: $u_1 \cdot u_2 \stackrel{\text{def}}{=} [\boldsymbol{v}_1 + \boldsymbol{v}_2, x_1 \cdot x_2]$ if $\boldsymbol{v}_1 + \boldsymbol{v}_2 \in \Lambda$, or $\perp$ otherwise, where $\perp$ means the operation is undefined and cannot be done.

We remark that $\cdot$ is only a partial binary operation and we use the following convention: $\perp + u = u + \perp = u \cdot \perp = \perp \cdot u = \perp$, for any $u \in \mathfrak{G} \cup \{\perp\}$. We then denote $\mathfrak{G}_{\boldsymbol{v}}$ the additive group $\{u = [\boldsymbol{v}', x] \in \mathfrak{G} \mid \boldsymbol{v}' = \boldsymbol{v}\}$. We will make natural use of vector and matrix operations over graded ring elements.

Cyclic Groups and Pairing-Friendly Settings. In the sequel, we consider graded rings over $R = \mathbb{Z}_p$ only, because we will use the vectorial space structure over $\mathbb{Z}_p^\ell$ in the proof of the smoothness of our generic construction of SPHF (see Section 4.2). This means we cannot directly deal with constructions in [GGH12] yet. Nevertheless, graded rings enable to easily deal with a cyclic group $\mathbb{G}$ of prime order p. In this case, $\kappa = \tau = 1$: elements $[0, x]$ of index 0 correspond to scalars $x \in \mathbb{Z}_p$ and elements $[1, x]$ of index 1 correspond to group elements $g^x \in \mathbb{G}$. We will see in the next section and in the Appendix D.1 that graded rings enable to deal with symmetric ($\kappa = 2, \tau = 1$) and asymmetric ($\kappa = 1, \tau = 2$) bilinear groups.

When we deal with generic constructions and generic graded rings, we use additive notation for all elements, because it is a lot easier. In concrete constructions, we still use multiplicative notation for group elements (and obviously additive notations for scalars $x \in \mathbb{Z}_p$). In bilinear groups (more formally defined in the Appendix D.1), 0 will be used to denote $0 \in \mathbb{Z}_p$, $1 = g^0 \in \mathbb{G}$ or $1 = e(g_1, g_2)^0 \in \mathbb{G}_T$, which will be clear from the context. However, we will sometimes write $1_\mathbb{G}$ for $1 = g^0 \in \mathbb{G}$, to avoid confusion with $1 \in \mathbb{Z}_p$. We will also use pairing notation $e(\cdot, \cdot)$ when going at a higher level in the indexes, as usual.

4.2 Generic Framework for GL-SPHF/KV-SPHF

In this section, we exhibit a generic framework for SPHF for languages of ciphertexts. We assume that crs is fixed and $\mathcal{S}et = \mathfrak{G}^\ell$ (*i.e.*, any word C is a tuple of ℓ graded ring elements), and we write $C = \boldsymbol{u} = (u_1, \ldots, u_\ell) \in \mathrm{L}_{\mathsf{aux}} = \mathrm{LOFC}_{\mathsf{full\text{-}aux}} \subseteq \mathfrak{G}^\ell$, where $\mathsf{full\text{-}aux} = (\mathsf{crs}, \mathsf{aux})$.

Language Representation. For a language $\mathrm{L}_{\mathsf{aux}}$, we assume there exist two positive integers k and n, a function $\Gamma : \mathfrak{G}^\ell \mapsto \mathfrak{G}^{k\times n}$, and a family of functions $\Theta_{\mathsf{aux}} : \mathfrak{G}^\ell \mapsto \mathfrak{G}^{1\times n}$, such that for any word $\boldsymbol{u} \in \mathcal{S}et$, $(\boldsymbol{u} \in \mathrm{L}_{\mathsf{aux}}) \iff (\exists \boldsymbol{\lambda} \in \mathfrak{G}^{1\times k}$ such that $\Theta_{\mathsf{aux}}(\boldsymbol{u}) = \boldsymbol{\lambda} \cdot \Gamma(\boldsymbol{u}))$. If Γ is a constant function (independent of the word $\boldsymbol{u}$), this defines a $\mathsf{KV\text{-}SPHF}$, otherwise this is a $\mathsf{GL\text{-}SPHF}$. However, in any case, we need the indexes of the components of $\Gamma(\boldsymbol{u})$ to be independent of $\boldsymbol{u}$.

We furthermore require that a user, who knows a witness w of the membership $\boldsymbol{u} \in \mathrm{L}_{\mathsf{aux}}$, can efficiently compute $\boldsymbol{\lambda}$. This may seem a quite strong requirement but, as shown in Section 5 and in the Appendix D, this is actually verified by very expressive languages over ciphertexts such as ElGamal, Cramer-Shoup and variants. We briefly illustrate it on our $\mathsf{KV\text{-}SPHF}$ on CS: $C = (u_1 = g_1^r, u_2 = g_2^r, e = M \cdot h^r, v = (cd^\xi)^r)$

$$\begin{array}{ll} \ell = 4 & \mathsf{aux} = M \\ k = 2 & n = 5 \end{array} \qquad \Gamma = \begin{pmatrix} g_1 & 0 & g_2 & h & c \\ 0 & g_1 & 0 & 0 & d \end{pmatrix} \qquad \boldsymbol{\lambda} = (r, r\xi) \qquad \begin{array}{c} \boldsymbol{\lambda}\cdot\Gamma = (g_1^r, g_1^{r\xi}, g_2^r, h^r, (cd^\xi)^r) \\ \Theta_M(C) = (u_1, u_1^\xi, u_2, e/M, v) \end{array}$$

Essentially, one tries to make the first columns of Γ, and the first components of Θ_{aux} to completely determine $\boldsymbol{\lambda}$ with some values that depend on the witnesses, which is the case with $u_1 = g_1^r$ and $u_1^\xi = g_1^{r\xi}$. The last columns of Γ and the last components of Θ_{aux} should define the language: we want $u_2 = g_2^r$, $e/M = h^r$, and $v = (cd^\xi)^r$, with the same r as for u_1. Some optimizations can then be applied.

Smooth Projective Hash Function. With the above notations, the hashing key is a vector $\mathsf{hk} = \boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^\intercal \stackrel{\$}{\leftarrow} \mathbb{Z}_p^n$, while the projection key is, for a word $\boldsymbol{u}$, $\mathsf{hp} = \boldsymbol{\gamma}(\boldsymbol{u}) = \Gamma(\boldsymbol{u}) \cdot \boldsymbol{\alpha} \in \mathfrak{G}^k$ (if Γ depends on $\boldsymbol{u}$). Then, the hash value is:

$$\mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, \boldsymbol{u}) \stackrel{\mathsf{def}}{=} \Theta_{\mathsf{aux}}(\boldsymbol{u}) \cdot \boldsymbol{\alpha} = \boldsymbol{\lambda} \cdot \boldsymbol{\gamma}(\boldsymbol{u}) \stackrel{\mathsf{def}}{=} \mathsf{ProjHash}(\mathsf{hp}, \mathsf{full\text{-}aux}, \boldsymbol{u}, w).$$

The set Π of hash values is exactly $\mathfrak{G}_{\boldsymbol{v}_H}$, the set of graded elements of index $\boldsymbol{v}_H$, the maximal index of the elements of $\Theta_{\mathsf{aux}}(\boldsymbol{u})$.

Our above Γ, $\boldsymbol{\lambda}$, and Θ_M immediately lead to our $\mathsf{KV\text{-}SPHF}$ on CS from the Section 2.4: with $\mathsf{hk} = (\eta_1, \eta_2, \theta, \lambda, \mu) \stackrel{\$}{\leftarrow} \mathbb{Z}_p^5$, the product with Γ leads to: $\mathsf{hp} = (\mathsf{hp}_1 = g_1^{\eta_1} g_2^\theta h^\lambda c^\mu, \mathsf{hp}_2 = g_1^{\eta_2} d^\mu) \in \mathbb{G}^2$, and

$$H = \mathsf{Hash}(\mathsf{hk}, (\mathsf{ek}, m), C) \stackrel{\mathsf{def}}{=} u_1^{(\eta_1+\xi\eta_2)} u_2^\theta (e/\mathcal{G}(m))^\lambda v^\mu = (\mathsf{hp}_1 \mathsf{hp}_2^\xi)^r \stackrel{\mathsf{def}}{=} \mathsf{ProjHash}(\mathsf{hp}, (\mathsf{ek}, m), C, r) = H'.$$

In addition, the following security analysis proves that the above generic SPHF is perfectly smooth, and thus proves the Theorem 2 as a particular case. We insist that if Γ really depends on $\boldsymbol{u}$ this construction yields a $\mathsf{GL\text{-}SPHF}$, whereas when Γ is a constant matrix, we obtain a $\mathsf{KV\text{-}SPHF}$, but perfectly smooth in both cases.

Security Analysis. In order to prove the smoothness of the above SPHF, we consider a word $\boldsymbol{u} \notin \mathrm{L}_{\mathsf{aux}}$ and a projection key $\mathsf{hp} = \boldsymbol{\gamma}(\boldsymbol{u}) = \Gamma(\boldsymbol{u}) \cdot \boldsymbol{\alpha}$: $\forall \boldsymbol{\lambda} \in \mathfrak{G}^{1\times k}, \Theta_{\mathsf{aux}}(\boldsymbol{u}) \neq \boldsymbol{\lambda} \cdot \Gamma(\boldsymbol{u})$. Using the projection $\mathfrak{L} : \mathfrak{G} \to \mathbb{Z}_p; u = [\boldsymbol{v}, x] \mapsto x$, which can be seen as the discrete logarithm, and which can be applied component-wise on vectors and matrices, this means that $\mathfrak{L}(\Theta_{\mathsf{aux}}(\boldsymbol{u}))$ is linearly independent from the rows of $\mathfrak{L}(\Gamma(\boldsymbol{u}))$. As a consequence, since $\boldsymbol{\alpha}$ is uniformly random, $\mathfrak{L}(\Theta_{\mathsf{aux}}(\boldsymbol{u})) \cdot \boldsymbol{\alpha}$ is a random variable independent from $\mathfrak{L}(\boldsymbol{\gamma}(\boldsymbol{u})) = \mathfrak{L}(\Gamma(\boldsymbol{u})) \cdot \boldsymbol{\alpha}$, and so from $\mathsf{hp} = \boldsymbol{\gamma}(\boldsymbol{u})$, since the index of $\boldsymbol{\gamma}(\boldsymbol{u})$ is a constant and thus $\mathfrak{L}(\boldsymbol{\gamma}(\boldsymbol{u}))$ completely defines $\boldsymbol{\gamma}(\boldsymbol{u})$. Therefore, H is a uniform element of $\mathfrak{G}_{\boldsymbol{v}_H}$ given hp, aux and $\boldsymbol{u}$.

5 Concrete Constructions of SPHFs

We now present various instantiations of KV-SPHFs, in order to illustrate our general framework. We insist that the purpose of this section is to explain how one can use the framework to generate some SPHFs in several situations: with or without pairings (with some constants from aux moved into crs). A concrete and more practical equation for the blind Waters signature is presented in the next section.

We start with one equation of linear multi-exponentiations, which requires pairing in its basic form. We thereafter study several improvements: by moving some constants from aux to crs (which can make sense in some applications), we will eliminate pairings, and also improve communication and computational costs.

5.1 KV-SPHF for Linear Multi-Exponentiation Equations

For the sake of clarity, we focus on ElGamal ciphertexts, but explain how to handle Cramer-Shoup ciphertexts in the Section 5.2. We refer the reader to the Section 5.3 and the Appendix D for more expressive languages.

Notations. We work on possibly two different groups $\mathbb{G}_1$ and $\mathbb{G}_2$, of the same prime order p, generated by g_1 and g_2, respectively, with a possible bilinear map into $\mathbb{G}_T$. We assume the DDH assumption hold in both $\mathbb{G}_1$ and $\mathbb{G}_2$. We define ElGamal encryption schemes with encryption keys $\mathsf{ek}_1 = (g_1, h_1 = g_1^{x_1})$ and $\mathsf{ek}_2 = (g_2, h_2 = g_2^{x_2})$ on each group. We are interested in languages on the ciphertexts $C_{1,i} = (u_{1,i} = g_1^{r_{1,i}}, e_{1,i} = h_1^{r_{1,i}} \cdot X_i)$, for $X_1, \ldots, X_{n_1} \in \mathbb{G}_1$, and $C_{2,j} = (u_{2,j} = g_2^{r_{2,j}}, e_{2,j} = h_2^{r_{2,j}} \cdot g_2^{y_j})$, for $y_1, \ldots, y_{n_2} \in \mathbb{Z}_p$, such that:

$$\prod_{i=1}^{n_1} X_i^{a_i} \cdot \prod_{j=1}^{n_2} A_j^{y_j} = B, \text{ with } \begin{array}{l} \mathsf{crs} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathsf{ek}_1, \mathsf{ek}_2) \\ \mathsf{aux} = (a_1, \ldots, a_{n_1}, A_1, \ldots, A_{n_2}, B) \in \mathbb{Z}_p^{n_2} \times \mathbb{G}_1^{n_2+1}. \end{array} \tag{1}$$

In this section, i and j will always range from 1 to n_1 and from 1 to n_2 respectively in all the products $\prod_i$, $\prod_j$ and tuples $(\cdot)_i$, $(\cdot)_j$. We can define the following elements, and namely the constant (n_2+1, n_2+2)-matrix Γ:

$$\Gamma = \left(\begin{array}{c|ccc|c} g_1 & 0 & \ldots & 0 & h_1 \\ \hline 0 & g_2 & & 0 & h_2 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & g_2 & h_2 \end{array}\right) \qquad \begin{aligned} \Theta_{\mathsf{aux}}(\boldsymbol{C}) &= \left(\prod_i u_{1,i}^{a_i}, (e(A_j, u_{2,j}))_j, \prod_i e(e_{1,i}^{a_i}, g_2) \cdot \prod_j e(A_j, e_{2,j})/e(B, g_2)\right) \\ \boldsymbol{\lambda} &= (\textstyle\sum_i a_i r_{1,i}, (A_j^{r_{2,j}})_j) \\ \boldsymbol{\lambda} \cdot \Gamma &= \left(g_1^{\sum_i a_i r_{1,i}}, (e(A_j, g_2^{r_{2,j}}))_j, e(h_1^{\sum_i a_i r_{1,i}}, g_2) \cdot \prod_j e(A_j^{r_{2,j}}, h_2)\right) \end{aligned}$$

We recall that in the matrix, 0 means $[\boldsymbol{v}, 0]$ for the appropriate index $\boldsymbol{v}$, and thus $1_{\mathbb{G}_1} = g_1^0 \in \mathbb{G}_1$ in the first line and column, but $1_{\mathbb{G}_2} = g_2^0 \in \mathbb{G}_2$ in the diagonal block. In addition, in the product $\boldsymbol{\lambda} \cdot \Gamma$, when adding two elements, they are first lifted in the minimal common higher ring, and when multiplying two elements, we either make a simple exponentiation (scalar with a group element) or a pairing (two group elements from different groups).

Because of the diagonal blocks in Γ, $\boldsymbol{\lambda}$ is implied by all but last components of $\Theta_{\mathsf{aux}}(\boldsymbol{C})$, then the last column defines the relation: the last component of $\Theta_{\mathsf{aux}}(\boldsymbol{C})$ is $\prod_i e(h_1^{r_{1,i} a_i} X^{a_i}, g_2) \cdot \prod_j e(A_j, h_2^{r_{2,j}} g_2^{y_j})/e(B, g_2)$, which is equal to the last component of $\boldsymbol{\lambda} \cdot \Gamma$, multiplied by the expression below, that is equal to 1 if and only if the relation (1) is satisfied:

$$\prod\nolimits_i e(X^{a_i}, g_2) \cdot \prod\nolimits_j e(A_j, g_2^{y_j})/e(B, g_2) = e\left(\prod\nolimits_i X^{a_i} \cdot \prod\nolimits_j A_j^{y_j}/B, g_2\right).$$

It thus leads to the following KV-SPHF, with $\mathsf{hp}_1 = g_1^{\nu} h_1^{\lambda}$ and $(\mathsf{hp}_{2,j} = g_2^{\theta_j} h_2^{\lambda})_j$, for $\mathsf{hk} = (\nu, (\theta_j)_j, \lambda)$:

$$H = \prod\nolimits_i e((u_{1,i}^{\nu} e_{1,i}^{\lambda})^{a_i}, g_2) \cdot \prod\nolimits_j e(A_j, u_{2,j}^{\theta_j} e_{2,j}^{\lambda}) \cdot e(B^{-\lambda}, g_2) = e(\mathsf{hp}_1^{\sum_i a_i r_{1,i}}, g_2) \cdot \prod\nolimits_j e(A_j^{r_{2,j}}, \mathsf{hp}_{2,j}) = H'.$$

As a consequence, the ciphertexts and the projection keys (which have to be exchanged in a protocol) globally consist of $2n_1 + 1$ elements from $\mathbb{G}_1$ and $3n_2$ elements from $\mathbb{G}_2$, and pairings are required for the hash value.

Ciphertexts with Randomness Reuse. A first improvement consists in using multiple independent encryption keys in $\mathbb{G}_2$, $\mathsf{ek}_{2,j} = (g_2, h_{2,j} = g_2^{x_{2,j}})$, for $j = 1, \dots, n_2$. This allows to reuse the same random coins [BBS03]. We are interested in languages on the ciphertexts $(C_{1,i} = (u_{1,i} = g_1^{r_{1,i}}, e_{1,i} = h_1^{r_{1,i}} \cdot X_i))_i$, for $(X_i)_i \in \mathbb{G}_1^{n_1}$, with $(r_{1,i})_i \in \mathbb{Z}_p^{n_1}$, and $C_2 = (u_2 = g_2^{r_2}, (e_{2,j} = h_{2,j}^{r_2} \cdot g_2^{y_j})_j)$, for $(y_j)_j \in \mathbb{Z}_p^{n_2}$, with $r_2 \in \mathbb{Z}_p$, still satisfying the same relation (1). This improves on the length of the ciphertexts of the g^{y_i}'s, from $2n_2$ group elements in $\mathbb{G}_2$ to $n_2 + 1$ in $\mathbb{G}_2$. A similar KV-SPHF as before can be derived, just modifying the last column vector $(h_2)_j$ by $(h_{2,j})_j$. Globally, the ciphertexts and the projection keys consist of $2n_1 + 1$ elements from $\mathbb{G}_1$ and $2n_2 + 1$ elements from $\mathbb{G}_2$, but pairings are still required for the hash value.

Moving the $\boldsymbol{A_j}$ from aux to crs. When the constant group elements A_j are known in advance, *i.e.*, are in crs instead of aux, Γ can depend on them and we can choose:

$$\Gamma = \left(\begin{array}{c|c|ccc|c} g_1 & 0 & 0 & \dots & 0 & h_1 \\ \hline 0 & g_2 & h_{2,1} & \dots & h_{2,n_2} & 0 \\ \hline 0 & 0 & g_2 & & 0 & A_1^{-1} \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & & g_2 & A_{n_2}^{-1} \end{array}\right) \qquad \begin{aligned} \Theta_{\mathsf{aux}}(\boldsymbol{C}) &= \left(\prod_i u_{1,i}^{a_i}, u_2, (e_{2,j})_j, \prod_i e_{1,i}^{a_i}/B\right) \\ \boldsymbol{\lambda} &= (\textstyle\sum_i a_i r_{1,i}, r_2, (y_j)_j) \\ \boldsymbol{\lambda} \cdot \Gamma &= (g_1^{\sum_i a_i r_{1,i}}, g_2^{r_2}, (h_{2,j}^{r_2} g_2^{y_j})_j, h_1^{\sum_i a_i r_{1,i}} \cdot \textstyle\prod_j A_j^{-y_j}) \end{aligned}$$

where again, because of the diagonal blocks in Γ, $\boldsymbol{\lambda}$ is implied by all but last components of $\Theta_{\mathsf{aux}}(\boldsymbol{C})$. The last component of $\Theta_{\mathsf{aux}}(\boldsymbol{C})$ is then $\prod_i e_{1,i}^{a_i}/B = \prod_i h_1^{a_i r_{1,i}} X_i^{a_i}/B$ and thus equal to the last component of $\boldsymbol{\lambda} \cdot \Gamma$, multiplied by $\prod_i X_i^{a_i} \cdot \prod_j A_j^{y_j}/B$ that is equal to 1 if and only if the relation (1) is satisfied. One can note that no pairings are required anymore since in the product $\boldsymbol{\lambda} \cdot \Gamma$ one never has to "multiply" two group elements, but just a group element with a scalar (exponentiation): we can encrypt everything in $\mathbb{G}_1$, using $g_2 = g_1$, but still with independent encryption keys $(h_{2,j})_j$, all in the unique group $\mathbb{G}$. It thus leads to the following KV-SPHF, with $(\mathsf{hp}_1 = g_1^{\nu} h_1^{\lambda}$, $\mathsf{hp}_2 = g_2^{\gamma} \cdot \prod_j h_{2,j}^{\theta_j}$, and $(\mathsf{hp}_{3,j} = g_2^{\theta_j} A_j^{-\lambda})_j$, for $\mathsf{hk} = (\nu, \gamma, (\theta_j)_j, \lambda)$:

$$H = \prod\nolimits_i (u_{1,i}^{\nu} e_{1,i}^{\lambda})^{a_i} \cdot \prod\nolimits_j e_{2,j}^{\theta_j} \cdot u_2^{\gamma} \cdot B^{-\lambda} = \mathsf{hp}_1^{\sum_i a_i r_{1,i}} \cdot \mathsf{hp}_2^{r_2} \cdot \prod\nolimits_j \mathsf{hp}_{3,j}^{y_j} = H'.$$

Globally, the ciphertexts and the projection keys consist of $2n_1 + 2n_2 + 3$ elements from $\mathbb{G}$, but no pairings.

Moving all the constant values from aux to crs. In some cases, all the constant values, A_j and a_i can be known in advance and public. The matrix Γ can then exploit their knowledge. Since no pairings are needed anymore, we can also take all the elements in the same group $\mathbb{G}$. We use a common basis g, and apply the randomness-reuse technique for the whole ciphertext, for both $(X_i)_i$ and $(y_j)_j$, with independent encryption keys $(h_{1,i})_i$ and $(h_{2,j})_j$ in $\mathbb{G}$. A unique random r produces $u = g^r$, and $(e_{1,i})_i$ and $(e_{2,j})_j$. This reduces the length of the ciphertext to $n_1 + n_2 + 1$ group elements in $\mathbb{G}$:

$$\Gamma = \left(\begin{array}{c|ccc|c} g & h_{2,1} & \dots & h_{2,n_2} & \prod_i h_{1,i}^{a_i} \\ \hline 0 & g & & 0 & A_1^{-1} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & g & A_{n_2}^{-1} \end{array}\right) \qquad \begin{aligned} \Theta_{\mathsf{aux}}(\boldsymbol{C}) = \left(u, (e_{2,j})_j, \textstyle\prod_i e_{1,i}^{a_i}/B\right) \qquad \boldsymbol{\lambda} = (r, (y_j)_j) \\ \boldsymbol{\lambda} \cdot \Gamma = (g^r, (h_{2,j}^r g^{y_j})_j, \textstyle\prod_i h_{1,i}^{a_i r} \cdot \prod_j A_j^{-y_j}) \end{aligned}$$

Projection keys become more compact, with only $n_2 + 1$ group elements in $\mathbb{G}$: $\mathsf{hp}_1 = g_1^{\nu} \cdot \prod_j h_{2,j}^{\theta_j} \cdot (\prod_i h_{1,i}^{a_i})^{\lambda}$, and $(\mathsf{hp}_{2,j} = g^{\theta_j} A_j^{-\lambda})_j$, for $\mathsf{hk} = (\nu, (\theta_j)_j, \lambda)$: $H = u^{\nu} \cdot \prod_i e_{1,i}^{\lambda a_i} \cdot \prod_j e_{2,j}^{\theta_j} \cdot B^{-\lambda} = \mathsf{hp}_1^r \cdot \prod_j \mathsf{hp}_{2,j}^{y_j} = H'$. Globally, the ciphertexts and the projection keys consist of $n_1 + 2n_2 + 2$ elements from $\mathbb{G}$, with no pairings.

5.2 From ElGamal to Cramer-Shoup Encryption

In order to move from ElGamal ciphertexts to Cramer-Shoup ciphertexts, if one already has Γ, Θ_{aux} and $\boldsymbol{\Lambda}$, to guarantee that the ElGamal plaintexts satisfy a relation, one simply has to make a bigger matrix, diagonal per blocks, with blocks Γ and smallers $(\Gamma_k)_k$ for every ciphertext $(u_k, u'_k, e_k, v_k)_k$, where

$$\Gamma_k = \begin{pmatrix} g & 0 & g' & c \\ 0 & g & 0 & d \end{pmatrix} \qquad \boldsymbol{\lambda_k} = (r_k, r_k \xi_k) \qquad \begin{array}{c} \boldsymbol{\lambda}_k \cdot \Gamma_k = (g^{r_k}, g^{r_k \xi_k}, g'^{r_k}, (cd^{\xi_k})^{r_k}) \\ \Theta_M(C_k) = (u_k, u_k^{\xi_k}, u'_k, v_k) \end{array}$$

The initial matrix Γ guarantees the relations on the ElGamal pairs (u_k, e_k), and the matrices Γ_k add the internal relations on the Cramer-Shoup ciphertexts. In the worst case, hk is increased by $4n$ scalars and hp by $2n$ group elements, for n ciphertexts. But some more compact matrices can be obtained in many cases, with much shorter hashing and projection keys, by merging some lines or columns in the global matrix. But this is a case by case optimization.

5.3 Generalizations

If we except the two first ones, the SPHF constructions from this section are all done without requiring any pairing, but are still KV-SPHF, allowing us to handle non-quadratic multi-exponentiation equations without pairings. To further extend our new formalism we describe in the next section a concrete application to blind signatures (while with a GL-SPHF), and we present more languages in the Appendix D.

However, as above for Cramer-Shoup ciphertexts, if one wants to satisfy several equations at a time, one just has to first consider them independently and to make a global matrix with each sub-language-matrix in a block on the diagonal. The hashing keys and the projection keys are then concatenated, and the hash values are simply multiplied. Optimizations can be possible, as shown in the Appendix C for the SPHF involved in the blind signature.

6 Application to Blind Signatures

In [BPV12], the authors presented a technique to do efficient blind signatures using an SPHF: it is still the most efficient Waters blind signature known so far. Using our new techniques, we can get an even better construction: we first present a new GL-SPHF on bit encryption, and then show that the above randomness-reuse technique on the bit encryptions can be handled. For lack of space, a presentation of blind signatures and Waters blind signature is postponed to the Appendix C.

6.1 GL-SPHF on Bit Encryption

Let us consider an ElGamal ciphertext $C = (u = g^r, e = h^r g^y)$, in which one wants to prove that $y \in \{0, 1\}$. We can define the following matrix that depends on C, hence a GL-SPHF:

$$\Gamma(C) = \begin{pmatrix} g & h & 0 & 0 \\ 0 & g & u & e/g \\ 0 & 0 & g & h \end{pmatrix} \qquad \begin{array}{c} \Theta_{\mathsf{aux}}(C) = (u, e, 1_{\mathbb{G}}, 1_{\mathbb{G}}) \qquad \boldsymbol{\lambda} = (r, y, -ry) \\ \boldsymbol{\lambda} \cdot \Gamma(C) = (g^r, h^r g^y, (u/g^r)^y, (e/gh^r)^y) \end{array}$$

As already said, in this matrix, 0 means $[\boldsymbol{v}, 0]$ for the appropriate index $\boldsymbol{v}$, and thus $1_{\mathbb{G}} = g_1^0 \in \mathbb{G}$. However, we use $1_{\mathbb{G}}$ in $\Theta_{\mathsf{full\text{-}aux}}(C)$. Because of the triangular block in $\Gamma(C)$, one easily sees that $\Theta_{\mathsf{aux}}(\boldsymbol{C}) = \boldsymbol{\lambda} \cdot \Gamma(C)$ if and only if $g^{y(y-1)} = 1$, and thus that $y \in \{0, 1\}$. With $\mathsf{hp}_1 = g^\nu h^\theta$, $\mathsf{hp}_2 = g^\theta u^\eta (e/g)^\lambda$, and $\mathsf{hp}_3 = g^\eta h^\lambda$, for $\mathsf{hk} = (\nu, \theta, \eta, \lambda)$: $H = u^\nu e^\theta = \mathsf{hp}_1^r \cdot \mathsf{hp}_2^y / \mathsf{hp}_3^{ry} = H'$.

6.2 Two-Flow Waters Blind Signature

We refer the reader to [BPV12] for the notations and to the Appendix C for more details on the proof, and namely the construction of the GL-SPHF. Here, we give a sketch of the protocol with the communication cost:

– $\mathsf{Setup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates a pairing-friendly system $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, with g_1 and g_2 generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, a random generator $h_s \in \mathbb{G}_1$ as well as independent generators $\boldsymbol{u} = (u_i)_{i \in \{0,\ldots,\ell\}} \in \mathbb{G}_1^{\ell+1}$ for the Waters function, denoted below $\mathcal{F}(M) = u_0 \prod_i u_i^{M_i}$, for $M = (M_i)_i \in \{0,1\}^\ell$, $(x_i)_i \in \mathbb{Z}_p^\ell$ and sets $h_i = g_1^{x_i}$, for $i = 1, \ldots, \ell$. In the following, i will always range from 1 to ℓ. It sets $\mathsf{ek} = (h_i)_i$, while $g_s = \prod_i h_i$. It outputs the global $\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, \mathsf{ek}, g_s, h_s, \boldsymbol{u})$;
– $\mathsf{KeyGen}(\mathsf{param})$ picks at random $x \in \mathbb{Z}_p$, sets the signing key $\mathsf{sk} = h_s^x$ and the verification key $\mathsf{vk} = (g_s^x, g_2^x)$;
– $\mathsf{BSProtocol}\langle \mathcal{S}(\mathsf{sk}), \mathcal{U}(\mathsf{vk}, M)\rangle$ runs as follows, where $\mathcal{U}$ wants to get a signature on $M = (M_i)_i \in \{0,1\}^\ell$:
  - Message encryption: $\mathcal{U}$ chooses a random $r \in \mathbb{Z}_p$ and encrypts $u_i^{M_i}$ for all the i's with the same random r: $c_0 = g_1^r$ and $(c_i = h_i^r u_i^{M_i})_i$. $\mathcal{U}$ also encrypts vk_1^r, into $d_0 = g_1^s$, $d_1 = h_1^s \mathsf{vk}_1^r$, with a different random s: It eventually sends: $(c_0, (c_i)_i, (d_0, d_1))$, which consists of $\ell + 3$ $\mathbb{G}_1$-elements;
  - Signature Generation: $\mathcal{S}$ first computes $c = u_0 \prod_i c_i = (\prod_i h_i)^r \mathcal{F}(M) = g_s^r \mathcal{F}(M)$, and generates the blinded signature $(\sigma_1' = h_s^x c^t = h_s^x g_s^{rt} \mathcal{F}(M)^t, \sigma_2 = (g_s^t, g_2^t))$ for a random $t \stackrel{\$}{\leftarrow} \mathbb{Z}_p$;
  - SPHF: $\mathcal{S}$ checks that each (c_0, c_i) encrypts a bit and (d_0, d_1) encrypts the Diffie-Hellman value of $(g_1, c_0, \mathsf{vk}_1)$. With $\mathsf{hk} = (\eta, (\theta_i)_i, (\nu_i)_i, \gamma, (\mu_i)_i, \lambda)$ and $\mathsf{hp}_1 = g_1^\eta \cdot \prod_i h_i^{\theta_i} \cdot \mathsf{vk}_1^\lambda$, $(\mathsf{hp}_{2,i} = u_i^{\theta_i} c_0^{\nu_i} (c_i/u_i)^{\mu_i})_i$, $(\mathsf{hp}_{3,i} = g_1^{\theta_i} h_i^{\mu_i})_i$, and $\mathsf{hp}_4 = g_1^\gamma h_1^\lambda$: $H = c_0^\eta \cdot \prod_i c_i^{\theta_i} \cdot d_0^\gamma \cdot d_1^\lambda = \mathsf{hp}_1^r \cdot \prod_i \mathsf{hp}_{2,i}^{M_i} \cdot \mathsf{hp}_{3,i}^{-rM_i} \cdot \mathsf{hp}_4^s = H' \in \mathbb{G}_1$. This SPHF is easily obtained from the above $\mathsf{GL\text{-}SPHF}$ on bit encryption, as shown in the Appendix C;
  - Masked Signature: $\mathcal{S}$ sends $(\mathsf{hp}, \Sigma = \sigma_1' \times H, \sigma_2)$, which consists of $2\ell + 4$ $\mathbb{G}_1$-elements and 1 $\mathbb{G}_2$-element;
  - Signature Recovery: Upon receiving $(\mathsf{hp}, \Sigma, \sigma_2)$, using its witnesses and hp, $\mathcal{U}$ computes H' and unmasks σ_1'. Thanks to the knowledge of r, it can compute $\sigma_1 = \sigma_1' \times (\sigma_{2,1})^{-r}$. Note that if $H' = H$, then $\sigma_1 = h_s^x \mathcal{F}(M)^t$, which together with $\sigma_2 = (g_s^t, g_2^t)$ is a valid Waters signature on M;
– $\mathsf{Verif}(\mathsf{vk}, M, (\sigma_1, (\sigma_{2,1}, \sigma_{2,2}))$, checks whether $e(\sigma_1, g_2) = e(h, \mathsf{vk}_2) \cdot e(\mathcal{F}(M), \sigma_{2,2}) \wedge e(\sigma_{2,1}, g_2) = e(g_s, \sigma_{2,2})$.

The whole process requires only $3\ell + 7$ elements in $\mathbb{G}_1$ ($\ell + 3$ for the ciphertexts, $2\ell + 4$ for the projection key, Σ and $\sigma_{2,1}$) and 1 in $\mathbb{G}_2$ ($\sigma_{2,2}$). This is more efficient than the instantiation from [BPV12] ($5\ell + 6$ elements in $\mathbb{G}_1$ and 1 in $\mathbb{G}_2$) already using an SPHF, and much more efficient than the instantiation from [BFPV11] ($6\ell + 7$ elements in $\mathbb{G}_1$ and $6\ell + 5$ in $\mathbb{G}_2$) using a Groth-Sahai [GS08] NIZK.

References

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

[BBC+12] Fabrice Ben Hamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. Efficient uc-secure authenticated key-exchange for algebraic languages. Cryptology ePrint Archive, Report 2012/284, 2012.

[BBS03] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. Randomness re-use in multi-recipient encryption schemes. In *PKC 2003*, *LNCS* 2567, pages 85–99. Springer, January 2003.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO 2004*, *LNCS* 3152, pages 41–55. Springer, August 2004.

[BFPV11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In *PKC 2011*, *LNCS* 6571, pages 403–422. Springer, March 2011.

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

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

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

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

[Cha83] David Chaum. Blind signatures for untraceable payments. In *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1983.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT 2005*, *LNCS* 3494, pages 404–421. Springer, May 2005.

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

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

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

[GGH12] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices and applications. Cryptology ePrint Archive, Report 2012/610, 2012.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In *EUROCRYPT 2003*, *LNCS* 2656, pages 524–543. Springer, May 2003.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT 2008*, *LNCS* 4965, pages 415–432. Springer, April 2008.

[HK08] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. In *CRYPTO 2008*, *LNCS* 5157, pages 21–38. Springer, August 2008.

[HKKL07] Carmit Hazay, Jonathan Katz, Chiu-Yuen Koo, and Yehuda Lindell. Concurrently-secure blind signatures without random oracles or setup assumptions. In *TCC 2007*, *LNCS* 4392, pages 323–341. Springer, February 2007.

[JR12] Charanjit S. Jutla and Arnab Roy. Relatively-sound NIZKs and password-based key-exchange. In *PKC 2012*, *LNCS* 7293, pages 485–503. Springer, May 2012.

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

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

[LY12] Benoît Libert and Moti Yung. Non-interactive CCA-secure threshold cryptosystems with adaptive security: New framework and constructions. In *TCC 2012*, *LNCS* 7194, pages 75–93. Springer, March 2012.

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

[Poi12] David Pointcheval. Password-based authenticated key exchange (invited talk). In *PKC 2012*, *LNCS* 7293, pages 390–397. Springer, May 2012.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT 2005*, *LNCS* 3494, pages 114–127. Springer, May 2005.

A Preliminaries

A.1 Formal Definitions of the Basic Primitives

We first recall the definitions of some of the basic tools, with the corresponding security notions and their respective success/advantage.

Hash Function Family. A hash function family $\mathcal{H}$ is a family of functions $\mathfrak{H}_K$ from $\{0,1\}^*$ to a fixed-length output, either $\{0,1\}^{\mathfrak{K}}$ or $\mathbb{Z}_p$. Such a family is said *collision-resistant* if for any adversary $\mathcal{A}$ on a random function $\mathfrak{H}_K \stackrel{\$}{\leftarrow} \mathcal{H}$, it is hard to find a collision. More precisely, we denote

$$\mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(\mathcal{A}) = \Pr[\mathfrak{H}_K \stackrel{\$}{\leftarrow} \mathcal{H}, (m_0, m_1) \leftarrow \mathcal{A}(\mathfrak{H}_K) : \mathfrak{H}_K(m_0) = \mathfrak{H}_K(m_1)], \qquad \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) = \max_{\mathcal{A} \leq t}\{\mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(\mathcal{A})\}.$$

Labeled Encryption Scheme. A labeled public-key encryption scheme $\mathcal{E}$ is defined by four algorithms:

- $\mathsf{Setup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}(\mathsf{param})$ generates a pair of keys, the encryption key ek and the decryption key dk;
- $\mathsf{Encrypt}(\ell, \mathsf{ek}, m; r)$ produces a ciphertext c on the input message $m \in \mathcal{M}$ under the label ℓ and encryption key ek, using the random coins r;
- $\mathsf{Decrypt}(\ell, \mathsf{dk}, c)$ outputs the plaintext m encrypted in c under the label ℓ, or $\perp$ for an invalid ciphertext.

An encryption scheme $\mathcal{E}$ should satisfy the following properties

- *Correctness*: for all key pair $(\mathsf{ek}, \mathsf{dk})$, any label ℓ, all random coins r and all messages m,

$$\mathsf{Decrypt}(\ell, \mathsf{dk}, \mathsf{Encrypt}(\ell, \mathsf{ek}, m; r)) = m.$$

– *Indistinguishability under chosen-ciphertext attacks*: this security notion can be formalized by the following security game, where the adversary $\mathcal{A}$ keeps some internal state between the various calls FIND and GUESS, and makes use of the oracle ODecrypt:

- ODecrypt(ℓ, c): This oracle outputs the decryption of c under the label ℓ and the challenge decryption key dk. The input queries (ℓ, c) are added to the list $\mathcal{CT}$.

$\mathsf{Exp}^{\mathsf{ind\text{-}cca}-b}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
3. $(\ell^*, m_0, m_1) \leftarrow \mathcal{A}(\mathtt{FIND} : \mathsf{ek}, \mathsf{ODecrypt}(\cdot,\cdot))$
4. $c^* \leftarrow \mathsf{Encrypt}(\ell^*, \mathsf{ek}, m_b)$
5. $b' \leftarrow \mathcal{A}(\mathtt{GUESS} : c^*, \mathsf{ODecrypt}(\cdot,\cdot))$
6. IF $(\ell^*, c^*) \in \mathcal{CT}$ RETURN 0
7. ELSE RETURN b'

The advantages are

$$\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathcal{E}}(\mathcal{A}) = \Pr[\mathsf{Exp}^{\mathsf{ind\text{-}cca}-1}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind\text{-}cca}-0}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}) = 1] \quad \mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathcal{E}}(t) = \max_{\mathcal{A} \le t}\{\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathcal{E}}(\mathcal{A})\}.$$

A.2 Statistical and Computational Distances

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two probability distributions over a finite set $\mathcal{S}$ and let X and Y be two random variables with these two respective distributions.

Statistical Distance. The statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is also the statistical distance between X and Y:

$$\mathrm{Dist}(\mathcal{D}_1, \mathcal{D}_2) = \mathrm{Dist}(X, Y) = \sum_{x \in \mathcal{S}} |\Pr[\, X = x\,] - \Pr[\, Y = x\,]|\,.$$

If the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is less than or equal to ε, we say that $\mathcal{D}_1$ and $\mathcal{D}_2$ are ε-close or are ε-statistically indistinguishable. If the $\mathcal{D}_1$ and D_2 are 0-close, we say that $\mathcal{D}_1$ and $\mathcal{D}_2$ are perfectly indistinguishable.

Computational Distance. We say that $\mathcal{D}_1$ and $\mathcal{D}_2$ are (t, ε)-computationally indistinguishable, if, for every probabilistic algorithm $\mathcal{A}$ running in time at most t:

$$|\Pr[\, \mathcal{A}(X) = 1\,] - \Pr[\, \mathcal{A}(Y) = 1\,]| \le \varepsilon.$$

We can note that for any t and ε, $\mathcal{D}_1$ and $\mathcal{D}_2$ are (t, ε)-computationally indistinguishable, if they are ε-close.

A.3 Concrete Instantiations

All the analyses in this paper could be instantiated with ElGamal-like schemes, based on either the Decisional Diffie-Hellman (DDH) assumption, or the Decisional Linear (DLin) assumption. But we focus on the former only:

Definition 5 (Decisional Diffie-Hellman (DDH)). *The Decisional Diffie-Hellman assumption says that, in a group $(p, \mathbb{G}, g)$, when we are given (g^a, g^b, g^c) for unknown random $a, b \xleftarrow{\$} \mathbb{Z}_p$, it is hard to decide whether $c = ab \bmod p$ (a DH tuple) or $c \xleftarrow{\$} \mathbb{Z}_p$ (a random tuple). We define by $\mathsf{Adv}^{\mathsf{ddh}}_{p,\mathbb{G},g}(t)$ the best advantage an adversary can have in distinguishing a DH tuple from a random tuple within time t.*

Cramer-Shoup (CS) Encryption Scheme [CS98]: it can be turned into a labeled public-key encryption scheme:

– $\mathsf{Setup}(1^{\mathfrak{K}})$ generates a group $\mathbb{G}$ of order p, with a generator g

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

This scheme is indistinguishable against chosen-ciphertext attacks, under the DDH assumption and the collision-resistance / universal one-wayness of the hash function $\mathcal{H}$.

B Security Proof for LAKE

B.1 Security Model

In this paper, we focus on efficiency and propose (in Section 3) an extension of the PAKE security model presented by Bellare-Pointcheval-Rogaway [BPR00] model for PAKE, between n players in the presence of an adversary. The adversary $\mathcal{A}$ plays a find-then-guess game against n players $(P_i)_{i=1,\dots,n}$. It has access to several instances Π_U^s for each player $U \in \{P_i\}$ and can activate them (in order to model concurrent executions) via several queries, described below:

- $\mathsf{Execute}(U, s, U', t)$: one outputs the transcript of an execution of the protocol between the instance Π_U^s of U and the instance $\Pi_{U'}^t$ of U'. It models passive eavesdropping attacks;
- $\mathsf{Send}(U, s, U', t, m)$: one sends the message m to the instance $\Pi_{U'}^t$ of U' in the name of the instance Π_U^s of U. It models active attacks;
- $\mathsf{Reveal}(U, s)$: if the instance Π_U^s of U has "accepted", one outputs the session key, otherwise one outputs $\perp$. It models a possible bad later use of the session key;
- $\mathsf{Test}(U, s)$: one first flips a coin $b \stackrel{\$}{\leftarrow} \{0, 1\}$, if $b = 1$ one outputs $\mathsf{Reveal}(U, s)$, otherwise one outputs a truly random key. It models the secrecy of the session key.

We say that Π_U^s and $\Pi_{U'}^t$ have matching conversations if inputs-outputs of the former correspond to the outputs-inputs of the latter and vice-versa. They are then called *partners*. We say that an instance is *fresh* if the key exists and is not trivially known by the adversary: more precisely, Π_U^s is fresh if

- Π_U^s has *accepted* the session, which is required to compute a session key;
- Π_U^s has not been asked a Reveal-query;
- no $\Pi_{U'}^t$ with *matching conversations* with Π_U^s has been asked a Reveal-query.

A key exchange protocol is then said secure if keys are indistinguishable from random keys for adversaries. Formally, the adversary is allowed to ask as many Execute, Send and Reveal-queries as it likes, and then only one Test-query to a *fresh* instance Π_U^s of a player. The adversary wins if it has guessed correctly the bit b in this query.

B.2 Proof of Theorem 4

This proof follows the one from [KV11]. It starts from the real attack game, in a Game 0: $\mathsf{Adv}_0(\mathcal{A}) = \varepsilon$. We incrementally modify the simulation to make possible the trivial attacks only. In the first games, all the honest players have their own values, and the simulator knows and can use them. Following [KV11], we can assume that there are two kinds of Send-queries: $\mathsf{Send}_0(U, s, U')$-queries where the adversary asks the instance Π_U^s to initiate an execution with an instance of U'. It is answered by the flow U' should send to communicate with

U; $\mathsf{Send}_1(U, s, m)$-queries where the adversary sends the message m to the instance Π_U^s. It gives no answer back, but defines the session key, for possible later Reveal or Test-queries.

Game $\mathbf{G}_1$: We first modify the way $\mathsf{Execute}$-queries are answered: we replace C and C' by encryptions of a fixed message M_0, that parses as two private parts P and P' and a word W, such that W is not in the language induced by (pub, P). Since the hashing keys are known, the common session key is computed as

$$\mathsf{sk} = \mathsf{Hash}(\mathsf{hk}, ((\mathsf{ek}, \mathsf{pub}), priv'), C') \times \mathsf{Hash}(\mathsf{hk}', ((\mathsf{ek}, \mathsf{pub}), \mathsf{priv}), C).$$

Since we could have first modified the way to compute sk, that has no impact at all from the soundness of the SPHF, the unique difference comes from the different ciphertexts. This is anyway indistinguishable under the $\mathsf{IND\text{-}CPA}$ property of the encryption scheme, for each $\mathsf{Execute}$-query. Using a classical hybrid technique, one thus gets $|\mathsf{Adv}_1(\mathcal{A}) - \mathsf{Adv}_0(\mathcal{A})| \leq \mathsf{negl}()$.

Game $\mathbf{G}_2$: We modify again the way $\mathsf{Execute}$-queries are answered: we replace the common session key by a truly random value. Since the languages are not satisfied, the smoothness guarantees indistinguishability: $|\mathsf{Adv}_2(\mathcal{A}) - \mathsf{Adv}_1(\mathcal{A})| \leq \mathsf{negl}()$.

Game $\mathbf{G}_3$: We now modify the way one answers the Send_1-queries, by using a decryption oracle, or alternatively knowing the decryption key. More precisely, when a message (hp, C) is sent, three cases can appear:

- it has been generated (altered) by the adversary, then one first decrypts the ciphertext to get $(\mathsf{priv}', priv, W')$ used by the adversary. Then
  - If they are correct ($W' \in \mathrm{L}_{\mathsf{pub},\mathsf{priv}'}$) and consistent with the receiver's values ($\mathsf{priv}' = priv', priv = \mathsf{priv}$) —event Ev— one declares that $\mathcal{A}$ succeeds (saying that $b' = b$) and terminates the game;
  - if they are not both correct and consistent with the receiver's values, one chooses sk at random.
- it is a replay of a previous flow sent by the simulator, then, in particular, one knows the hashing keys, and one can compute the session keys using all the hashing keys.

The first case can only increase the advantage of the adversary in case Ev happens (which probability is computed in $\mathbf{G}_6$). The second change is indistinguishable under the adaptive-smoothness and thus only increases the advantage of the adversary by a negligible term. The third change does not affect the way the key is computed, so finally: $\mathsf{Adv}_2(\mathcal{A}) \leq \mathsf{Adv}_3(\mathcal{A}) + \mathsf{negl}()$.

Game $\mathbf{G}_4$: We modify again the way one answers the Send_1-queries. More precisely, when a message (hp, C) is sent, two cases can appear:

- if there is an instance $\Pi_{U'}^t$ partnered with Π_U^s that receives this flow, then set the key identical to the key for $\Pi_{U'}^t$;
- otherwise, one chooses sk at random.

The former case remains identical since the message is a replay of a previous flow, and the latter is indistinguishable, as in [KV11], thanks to the adaptive-smoothness and their technical lemma that proves that all the hash values are random looking even when hashing keys and ciphertexts are re-used: $|\mathsf{Adv}_4(\mathcal{A}) - \mathsf{Adv}_3(\mathcal{A})| \leq \mathsf{negl}()$.

Game $\mathbf{G}_5$: We now modify the way one answers the Send_0-queries: instead of encrypting the correct values, one does as in $\mathbf{G}_1$ for $\mathsf{Execute}$-queries, and encrypts M_0. Since for simulating the Send_1-queries decryptions are required, indistinguishability relies on the $\mathsf{IND\text{-}CCA}$ security of the encryption scheme: $|\mathsf{Adv}_5(\mathcal{A}) - \mathsf{Adv}_4(\mathcal{A})| \leq \mathsf{negl}()$.

Game $\mathbf{G}_6$: For all the hashing and projection keys, we now use the dummy private inputs. Since we restricted hk and hp not to depend on aux, the distributions of these keys are independent of the auxiliary private inputs: $|\mathsf{Adv}_6(\mathcal{A}) - \mathsf{Adv}_5(\mathcal{A})| \leq \mathsf{negl}()$.

If one combines all the relations, one gets $\mathsf{Adv}_6(\mathcal{A}) \geq \mathsf{Adv}_0(\mathcal{A}) - \mathsf{negl}() = \varepsilon - \mathsf{negl}()$.

One can note that in this final game, the values of the honest players are not used anymore during the simulation, but just for declaring whether the adversary has won or not (event Ev). Otherwise, non-partnered players have random and independent keys, and thus unless the simulator stops the simulation, the advantage in the last game is exactly 0: $\mathsf{Adv}_6(\mathcal{A}) = \Pr[\mathsf{Ev}]$. And thus, we have $\varepsilon \leq \Pr[\mathsf{Ev}] + \mathsf{negl}()$.

Let us recall that Ev means that the adversary has encrypted $(\mathsf{priv}', priv, W')$ that are correct ($W' \in \mathrm{L}_{\mathsf{pub},\mathsf{priv}'}$) and consistent with the receiver's values ($\mathsf{priv}' = priv', priv = \mathsf{priv}$). Since the values for the honest players are never used during the simulation, we can assume we choose them at the very end only to check whether event Ev happened:

$$\Pr[\mathsf{Ev}] = \Pr[\exists k : \mathsf{priv}'(k) = priv'_{i_k}, priv(k) = priv_{i_k}, W'(k) \in \mathrm{L}_{\mathsf{pub},\mathsf{priv}'_{i_k}}]$$

where k lists all the Send_1-queries with adversary-generated messages in which the ciphertexts decrypt to $(\mathsf{priv}'(k), priv(k), W'(k))$, and i_k is the index of the recipient of k-th Send_1-query: it has first to guess the private values, and then once it has guessed them it has to find a word in the language:

$$\Pr[\mathsf{Ev}] \leq \frac{q_s}{2^m} \times \mathsf{Succ}^{\mathrm{L}}(t),$$

where m is the minimal min-entropy on the joint distributions of the $(\mathsf{priv}, priv')$ for any two players U, U' who want to communicate, and $\mathsf{Succ}^{\mathrm{L}}(t)$ is the best success an adversary can get in finding a word in a language $\mathrm{L}_{\mathsf{pub},\mathsf{priv}}$. Then, by combining all the inequalities, one gets

$$\varepsilon \leq \frac{q_s}{2^m} \times \mathsf{Succ}^{\mathrm{L}}(t) + \mathsf{negl}().$$

C Security Proof for the Blind Signature

In this appendix, we give details on our two-flow Waters blind signature scheme outlined in Section 6. We first present the asymmetric variant of Waters signatures proposed in [BFPV11] and then recall the formal security definitions of blind signatures and of their security properties. Using our formalism, we describe in details the SPHF used in the scheme and finally prove the security of our scheme.

C.1 Waters Signature (Asymmetric Setting)

In 2011, Blazy, Fuchsbauer, Pointcheval and Vergnaud [BFPV11] proposed the following variant of Waters signatures in an asymmetric pairing-friendly environment:

- $\mathsf{Setup}(1^{\mathfrak{K}})$: in a pairing-friendly environment $(p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, \mathbb{G}_T, e)$, one chooses a random vector $\boldsymbol{u} = (u_0, \ldots, u_\ell) \stackrel{\$}{\leftarrow} \mathbb{G}_1^{\ell+1}$, and for convenience, we denote $\mathcal{F}(M) = u_0 \prod_{i=1}^{\ell} u_i^{M_i}$ for $M = (M_i)_i \in \{0,1\}^\ell$. We also need two extra generators $(g_s, h_s) \stackrel{\$}{\leftarrow} \mathbb{G}_1^2$. The global parameters param consist of all these elements $(p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, \mathbb{G}_T, e, g_s, h_s, \boldsymbol{u})$.
- $\mathsf{KeyGen}(\mathsf{param})$ chooses a random scalar $x \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, which defines the public key as $\mathsf{vk} = (g_s^x, g_2^x) = (\mathsf{vk}_1, \mathsf{vk}_2)$, and the secret key is set as $\mathsf{sk} = h_s^x$.
- $\mathsf{Sign}(\mathsf{sk}, M; s)$ outputs, for some random $t \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, $\sigma = \big(\sigma_1 = \mathsf{sk} \cdot \mathcal{F}(M)^t, \sigma_2 = (\sigma_{2,1} = g_s^t, \sigma_{2,2} = g_2^t)\big)$.
- $\mathsf{Verif}(\mathsf{vk}, M, \sigma)$ checks whether $e(\sigma_1, g_2) = e(h_s, \mathsf{vk}_2) \cdot e(\mathcal{F}(M), \sigma_{2,2})$, and $e(\sigma_{2,1}, g_2) = e(g_s, \sigma_{2,2})$.

This scheme is unforgeable against (adaptive) chosen-message attacks under the following variant of the CDH assumption, which states that CDH is hard in $\mathbb{G}_1$ when one of the random scalars is also given as an exponentiation in $\mathbb{G}_2$:

Definition 6 (The Advanced Computational Diffie-Hellman problem (CDH^+)). *In a pairing-friendly environment $(p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, \mathbb{G}_T, e)$. The CDH^+ assumption states that given $(g_1, g_2, g_1^a, g_2^a, g_1^b)$, for random $a, b \in \mathbb{Z}_p$, it is hard to compute g_1^{ab}.*

$\mathsf{Exp}^{\mathsf{bl}-b}_{\mathcal{BS},\mathcal{S}^*}(\mathfrak{K})$
1. $\mathsf{param} \leftarrow \mathsf{BSSetup}(1^{\mathfrak{K}})$
2. $(\mathsf{vk}, M_0, M_1) \leftarrow \mathcal{A}(\mathtt{FIND} : \mathsf{param})$
3. $\sigma_b \leftarrow \mathsf{BSProtocol}\langle \mathcal{A}, \mathcal{U}(\mathsf{vk}, M_b)\rangle$
4. $\sigma_{1-b} \leftarrow \mathsf{BSProtocol}\langle \mathcal{A}, \mathcal{U}(\mathsf{vk}, M_{1-b})\rangle$
5. $b^* \leftarrow \mathcal{S}^*(\mathtt{GUESS} : \sigma_0, \sigma_1)$;
6. RETURN $b^* = b$

Blindness property

$\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{BS},\mathcal{U}^*}(\mathfrak{K})$
1. $(\mathsf{param}) \leftarrow \mathsf{BSSetup}(1^{\mathfrak{K}})$
2. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{BSKeyGen}(\mathsf{param})$
3. For $i = 1, \ldots, q_s$, $\mathsf{BSProtocol}\langle \mathcal{S}(\mathsf{sk}), \mathcal{A}(\mathtt{INIT} : \mathsf{vk})\rangle$
4. $\big((m_1, \sigma_1), \ldots, (m_{q_s+1}, \sigma_{q_s+1})\big) \leftarrow \mathcal{A}(\mathtt{GUESS} : \mathsf{vk})$;
5. IF $\exists i \neq j, m_i = m_j$ OR $\exists i, \mathsf{Verif}(\mathsf{pk}, m_i, \sigma_i) = 0$ RETURN 0
6. ELSE RETURN 1

Unforgeability

Fig. 3. Security Games for $\mathcal{BS}$

C.2 Security Definitions for Blind Signatures

Blind signature schemes, introduced by Chaum in 1982 [Cha83], allow a person to get a signature by another party without revealing any information about the message being signed. A blind signature can then be publicly verified using the unblinded message.

Definition 7 (Blind Signature Scheme). *A blind signature scheme* $\mathcal{BS}$ *is defined by three algorithms* $(\mathsf{BSSetup}, \mathsf{BSKeyGen}, \mathsf{BSVerif})$ *and one interactive protocol* $\mathsf{BSProtocol}\langle \mathcal{S}, \mathcal{U}\rangle$*:*

- $\mathsf{BSSetup}(1^{\mathfrak{K}})$*, generates the global parameters* param *of the system;*
- $\mathsf{BSKeyGen}(\mathsf{param})$ *is a probabilistic polynomial-time algorithm that generates a pair of keys* $(\mathsf{vk}, \mathsf{sk})$ *where* vk *is the public (verifying) key and* sk *is the secret (signing) key;*
- $\mathsf{BSProtocol}\langle \mathcal{S}(\mathsf{sk}), \mathcal{U}(\mathsf{vk}, M)\rangle$*: this is a probabilistic polynomial-time interactive protocol between the algorithms* $\mathcal{S}(\mathsf{sk})$ *and* $\mathcal{U}(\mathsf{vk}, M)$*, for a message* $M \in \{0,1\}^n$*. It generates a signature* σ *on* M *under* vk *related to* sk *for the user.*
- $\mathsf{BSVerif}(\mathsf{vk}, M, \sigma)$ *is a deterministic polynomial-time algorithm which outputs 1 if the signature* σ *is valid with respect to* m *and* vk*, 0 otherwise.*

A blind signature scheme $\mathcal{BS}$ should satisfy the two following security notions: the blindness condition that is a guarantee for the signer, and the unforgeability that is a guarantee for the signer. The blindness states that a malicious signer should not be able to link the final signatures output by the user to the individual *valid* interactions with the user. We insist on *valid* executions which end with a valid signature σ of the message used by $\mathcal{U}$ under the key vk. The signer could of course send a wrong answer which would lead to an invalid signature in one execution of $\mathsf{BSProtocol}\langle \mathcal{S}(\mathsf{sk}), \mathcal{U}(\mathsf{vk}, M)\rangle$. Then, it could easily distinguish a valid signature from an invalid one, and thus valid execution of the protocol and the invalid one. However this malicious behaviour is a kind of denial of service and is out of scope of this work. Therefore, in this paper *blindness* formalizes that one valid execution is indistinguishable for the signer from other valid executions. This notion was formalized in [HKKL07] and termed *a posteriori blindness*. The unforgeability property insures that an adversary, interacting freely with an honest signer, should not be able to produce $q+1$ valid signatures after at most q complete interactions with the honest signer (for any q polynomial in the security parameter).

These security notions are precised by the security games presented on Figure 3, where the adversary keeps some internal state between the various calls INIT, FIND and GUESS.

C.3 Underlying SPHF in the Blind Signature Scheme

Following [BPV12], our scheme makes use of an SPHF in the interactive signing protocol to insure (in an efficient way) that the user actually knows the signed message. As outlined in Section 6, during the interactive process of the blind signature protocol, we have:

- General setting: a pairing-friendly system $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, with g_1 and g_2 generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively;

– Encryption parameters: random scalars $(x_i)_i \in \mathbb{Z}_p^\ell$ with $(h_i = g_1^{x_i})_i$, where i ranges from 1 to ℓ, as everywhere in the following. Then, $\mathsf{ek} = (h_i)_i$;
– Signature parameters: independent generators $\boldsymbol{u} = (u_i)_{i \in \{0,\ldots,\ell\}} \in \mathbb{G}_1^{\ell+1}$ for the Waters function, $g_s = \prod_i h_i$, and a random generator $h_s \in \mathbb{G}_1$, then $\mathsf{sk} = h_s^x$ and $\mathsf{vk} = (g_s^x, g_2^x)$, for a random scalar x.

The user has generated $c_0 = g_1^r$ and $c_i = h_i^r u_i^{M_i}$ for $i = 1,\ldots,\ell$, as well as $d_0 = g_1^s$, $d_1 = h_1^s \mathsf{vk}_1^r$. In the following simulation, we will extract $(M_i)_i$ from $C = (c_0, (c_i)_i)$, and we thus need to be sure that this message can be extracted. In addition, the simulator will also need to know vk_1^r to generate the blinded signature, hence its encryption in (d_0, d_1). But this has to be checked, with the following the language membership:

1. each (c_0, c_i) encrypts a bit;

$$\Gamma(C) = \left(\begin{array}{c|ccc|ccc|ccc}
g_1 & h_1 & \ldots & h_\ell & 0 & \ldots & 0 & 0 & \ldots & 0 \\
\hline
0 & u_1 & & 0 & c_0 & & 0 & c_1/u_1 & & 0 \\
\vdots & & \ddots & & & \ddots & & & \ddots & \\
0 & 0 & & u_\ell & 0 & & c_0 & 0 & & c_\ell/u_\ell \\
\hline
0 & & & & g_1 & & 0 & h_1 & & 0 \\
\vdots & & 0 & & & \ddots & & & \ddots & \\
0 & & & & 0 & & g_1 & 0 & & h_\ell
\end{array}\right) \qquad \begin{array}{c} \Theta_{\mathsf{aux}}(C) = (c_0, (c_i)_i, (1_{\mathbb{G}_1})_i, (1_{\mathbb{G}_1})_i) \\ \boldsymbol{\lambda} = (r, (M_i)_i, (-rM_i)_i) \end{array}$$

$$\boldsymbol{\lambda} \cdot \Gamma(C) = (g_1^r, (h_i^r u_i^{M_i})_i, (c_0^{M_i} g_1^{-rM_i})_i, ((c_i/u_i h_i^r)^{M_i})_i).$$

2. the ciphertext (d_0, d_1) encrypts the Diffie-Hellman value of $(g_1, c_0, \mathsf{vk}_1)$;

$$\Gamma = \begin{pmatrix} g_1 & 0 & \mathsf{vk}_1 \\ 0 & g_1 & h_1 \end{pmatrix} \qquad \begin{array}{c} \Theta_{\mathsf{aux}}(C) = (c_0, d_0, d_1) \\ \boldsymbol{\lambda} \cdot \Gamma = (g_1^r, g_1^s, \mathsf{vk}_1^r h_1^s) \end{array} \qquad \boldsymbol{\lambda} = (r, s)$$

The two matrices can be compressed with a common row/column: the same witness r is indeed used in both matrices, the two corresponding rows can be merged; the first column is the same in both matrices, it can thus be a common one:

$$\Gamma(C) = \left(\begin{array}{c|ccc|ccc|c|ccc|c}
g_1 & h_1 & \ldots & h_\ell & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & \mathsf{vk}_1 \\
\hline
0 & u_1 & & 0 & c_0 & & 0 & 0 & c_1/u_1 & & 0 & 0 \\
\vdots & & \ddots & & & \ddots & & \vdots & & \ddots & & \vdots \\
0 & 0 & & u_\ell & 0 & & c_0 & 0 & 0 & & c_\ell/u_\ell & 0 \\
\hline
0 & & & & g_1 & & 0 & 0 & h_1 & & 0 & 0 \\
\vdots & & 0 & & & \ddots & & \vdots & & \ddots & & \vdots \\
0 & & & & 0 & & g_1 & 0 & 0 & & h_\ell & 0 \\
\hline
0 & 0 & \ldots & 0 & 0 & \ldots & 0 & g_1 & 0 & \ldots & 0 & h_1
\end{array}\right) \qquad \begin{array}{c} \Theta_{\mathsf{aux}}(C) = (c_0, (c_i)_i, (1_{\mathbb{G}_1})_i, d_0, (1_{\mathbb{G}_1})_i, d_1) \\ \boldsymbol{\lambda} = (r, (M_i)_i, (-rM_i)_i, s) \end{array}$$

$$\boldsymbol{\lambda} \cdot \Gamma(C) = (g_1^r, (h_i^r u_i^{M_i})_i, (c_0^{M_i} g_1^{-rM_i})_i, g_1^s, ((c_i/u_i h_i^r)^{M_i})_i, \mathsf{vk}_1^r h_1^s).$$

This leads to, with $\mathsf{hk} = (\eta, \{\theta_i\}_i, \{\nu_i\}_i, \gamma, \{\mu_i\}_i, \lambda)$,

$$\mathsf{hp}_1 = g_1^\eta \cdot \prod\nolimits_i h_i^{\theta_i} \cdot \mathsf{vk}_1^\lambda \qquad (\mathsf{hp}_{2,i} = u_i^{\theta_i} c_0^{\nu_i} (c_i/u_i)^{\mu_i})_i \qquad (\mathsf{hp}_{3,i} = g_1^{\nu_i} h_i^{\mu_i})_i \qquad \mathsf{hp}_4 = g_1^\gamma h_1^\lambda$$

$$H = c_0^\eta \cdot \prod\nolimits_i c_i^{\theta_i} \cdot d_0^\gamma \cdot d_1^\lambda = \mathsf{hp}_1^r \cdot \prod\nolimits_i \mathsf{hp}_{2,i}^{M_i} \cdot \mathsf{hp}_{3,i}^{-rM_i} \cdot \mathsf{hp}_4^s = H'.$$

The signers thus uses H to mask his blinded signature (σ_1', σ_2). But since σ_2 is just a random pair, only σ_1' needs to be masked. Without it, one cannot forge a signature, but it can be unmasked by the user with H', if the values $(c_0, (c_i)_i, (d_0, d_1))$ are in the correct language, and thus are correct ciphertexts.

One can note that the projection key consists of $2\ell + 2$ group elements in $\mathbb{G}_1$, and the hash value is in $\mathbb{G}_1$. No pairings are needed for this SPHF. Since Γ depends on C, this is a $\mathsf{GL\text{-}SPHF}$, but this is enough for our interactive protocol.

C.4 Security proofs

Proposition 8. *Our blind signature scheme is blind under the* DDH *assumption in* $\mathbb{G}_1$[1]*:*

$$\mathsf{Adv}^{\mathsf{bl}}_{\mathcal{BS},\mathcal{A}}(\mathfrak{K}) \leq 2 \times (\ell + 1) \times \mathsf{Adv}^{\mathsf{DDH}}_{p,\mathbb{G}_1,g_1}(\mathfrak{K}).$$

Proof. Let us consider an adversary $\mathcal{A}$ against the blindness of our scheme. We build an adversary $\mathcal{B}$ against the DDH assumption in $\mathbb{G}_1$.

Game $\mathbf{G}_0$: In a first game $\mathbf{G}_0$, we run the standard protocol:

- $\mathsf{BSSetup}(1^k)$, $\mathcal{B}$ generates $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ with g_1 and g_2 generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. It also generates independent generators $\boldsymbol{u} = (u_i)_{i\in\{0,\ldots,\ell\}} \in \mathbb{G}_1^{\ell+1}$ for the Waters function and sets $\mathsf{ek} = (h_i)_i$ and $g_s = \prod_i h_i$. It generates $h_s = g_s^\alpha \in \mathbb{G}_1$ and defines the global parameters as $\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, \mathsf{ek}, g_s, h_s, \boldsymbol{u})$;
- The adversary $\mathcal{A}$ generates a verification key $\mathsf{vk} = (\mathsf{vk}_1, \mathsf{vk}_2) \in \mathbb{G}_1 \times \mathbb{G}_2$ such that $e(\mathsf{vk}_1, g_2) = e(g_s, \mathsf{vk}_2)$ and two ℓ-bit messages M^0, M^1.
- $\mathcal{A}$ and $\mathcal{B}$ run twice the interactive issuing protocol, first on the message M^b, and then on the message M^{1-b}:
  - $\mathcal{B}$ chooses a random $r \in \mathbb{Z}_p$ and encrypts $u_i^{M_i}$ for all the i's with the same random r: $c_0 = g_1^r$ and $(c_i = h_i^r u_i^{M_i^b})_i$. $\mathcal{B}$ also encrypts vk_1^r, into $d_0 = g_1^s$, $d_1 = h_1^s \mathsf{vk}_1^r$ and sends $(c_0, (c_i)_i, (d_0, d_1))$ to $\mathcal{A}$.
  - $\mathcal{A}$ then outputs $(\mathsf{hp}, \Sigma = \sigma_1' \times H, \sigma_2)$
  - $\mathcal{B}$, using its witnesses and hp, computes H' and unmasks $\sigma_1' = \Sigma/H$ which together with σ_2 should be a valid Waters Signature on M^b. It then randomizes the signature with s' to get Σ_b.

  The same is done a second time with M^{1-b} to get Σ_{1-b}.
- $\mathcal{B}$ publishes (Σ_0, Σ_1).
- Eventually, $\mathcal{A}$ outputs b'.

We denote by ε the advantage of $\mathcal{A}$ in this game. By definition, we have:

$$\varepsilon = \mathsf{Adv}^{\mathsf{bl}}_{\mathcal{BS},\mathcal{A}}(k) = \Pr_{\mathbf{G}_0}[b' = 1 | b = 1] - \Pr_{\mathbf{G}_0}[b' = 1 | b = 0] = 2 \times \Pr_{\mathbf{G}_0}[b' = b] - 1.$$

Game $\mathbf{G}_1$: In a second game $\mathbf{G}_1$, we modify the way $\mathcal{B}$ extracts the signatures Σ_b and Σ_{1-b}. Since $\mathcal{B}$ knows the scalar α such that $h_s = g_s^\alpha$ it can compute the secret key $\mathsf{sk} = \mathsf{vk}_1^\alpha$ associated to $\mathsf{vk} = (\mathsf{vk}_1, \mathsf{vk}_2)$. One can note that, since we focus on valid executions with the signer, and due to the re-randomization of Waters signatures which leads to random signatures, $\mathcal{B}$ can generates itself random signatures on M^b and M^{1-b} using sk. This game is perfectly indistinguishable from the previous one:

$$\Pr_{\mathbf{G}_1}[b' = b] = \Pr_{\mathbf{G}_0}[b' = b].$$

Game $\mathbf{G}_2$: In this final game, we replace all the ciphertexts sent by $\mathcal{B}$ by encryption of random group elements in $\mathbb{G}_1$. For proving indistinguishability with the previous game, we use the hybrid technique for ElGamal ciphertexts with randomness re-use [BBS03]:

$$\varepsilon \leq 2 \times (\ell + 1) \times \mathsf{Adv}^{\mathsf{DDH}}_{p,\mathbb{G}_1,g_1}(\mathfrak{K}) + 2 \times \Pr_{\mathbf{G}_2}[b' = b] - 1.$$

In this last game, the two executions are thus perfectly indistinguishable, and thus $\Pr_{\mathbf{G}_2}[b' = b] = 1/2$ and we get the bound claimed in the proposition. □

[1] This assumption is sometimes referred to as the XDH assumption.

Proposition 9. *Our blind signature scheme is unforgeable under the* CDH^+ *assumption.*

$$\mathsf{Adv}^{\mathsf{uf}}_{\mathcal{BS},\mathcal{A}}(\mathfrak{K}) \leq \Theta\left(\frac{\mathsf{Succ}^{\mathsf{CDH}^+}_{p,\mathbb{G}_1,g_1,\mathbb{G}_2,g_2}(\mathfrak{K})}{q_s\sqrt{\ell}}\right).$$

Proof. Let $\mathcal{A}$ be an adversary against the Unforgeability of the scheme. We assume that this adversary is able after q_s signing queries to output at least q_s+1 valid signatures on different messages (for some q_s polynomial in the security parameter). We now build an adversary $\mathcal{B}$ against the CDH^+ assumption.

- $\mathcal{B}$ is first given a CDH^+ challenge $(g_s, g_2, g_s^x, g_2^x, h_s)$ in a pairing-friendly environment $(p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, \mathbb{G}_T, e)$
- $\mathcal{B}$ emulates $\mathsf{BSSetup}$: it picks a random position $j \stackrel{\$}{\leftarrow} \{0,\dots,\ell\}$, random indices $y_0,\dots,y_\ell \stackrel{\$}{\leftarrow} \{0,\dots,2q_s-1\}$ and random scalars $z_0,\dots,z_\ell \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and publishes $\boldsymbol{u} = (u_i)_{i\in\{0,\dots,\ell\}} \in \mathbb{G}^{\ell+1}$ for the Waters function, where $u_0 = h_s^{y_0-2jq_s} g_s^{z_0}$ and $u_i = h_s^{y_i} g_s^{z_i}$ for $i \in \{1,\dots,\ell\}$. It sets $g_1 = g_s^\gamma$ and $\mathsf{ek} = (h_i)_i$ with $h_i = g_1^{a_i} \in \mathbb{G}_1$ for $i \in \{1,\dots,\ell\}$ for some known random scalars $a_1,\dots,a_\ell$ and $\gamma = 1/\sum_i a_i \bmod p$. It keeps secret the associated decryption key $\mathsf{dk} = (a_1,\dots,a_\ell) \in \mathbb{Z}_p^\ell$ and outputs the global $\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, \mathsf{ek}, g_s, h_s, \boldsymbol{u})$.
- $\mathcal{B}$ then emulates $\mathsf{BSKeyGen}$: it publishes $\mathsf{vk} = (g_s^x, g_2^x)$ from the challenge as its verification key (one can note that recovering the signing key h_s^x is the goal of our adversary $\mathcal{B}$);
- $\mathcal{A}$ can now interact q_s times with the signer, playing the interactive protocol $\mathsf{BSProtocol}\langle\mathcal{S},\mathcal{A}\rangle$:
  - $\mathcal{A}$ sends the bit-per-bit encryptions c_i for $i \in \{1,\dots,\ell\}$, and the extra ciphertext (d_0, d_1) hiding Y the verification key vk_1 raised to the randomness;
  - Thanks to dk, $\mathcal{B}$ is able to extract M from the bit-per-bit ciphertexts (either the decryption leads to u_i and so $M_i = 1$, or to g_1 and so $M_i = 0$), and $Y = \mathsf{vk}_1^r$ from the additional ciphertext (d_0, d_1). One can also compute $c_0^{1/\gamma} = g_s^r$.
  - If one of the extracted terms is not of the right form (either not a bit in the c_i, or $(g_s, g_s^r, \mathsf{vk}_1, Y)$ is not a Diffie-Hellman tuple, which occurs if $e(g_s^r, \mathsf{vk}_2) \neq e(Y, g_2)$ and can thus be checked with a pairing computation), then $\mathcal{A}$ has submitted a "word" not in the appropriate language for the SPHF. Therefore through the smoothness property of the SPHF, it is impossible from a theoretic point of view that the adversary extracts anything from $\mathcal{B}$'s answer, therefore $\mathcal{B}$ simply sends a random element Σ in $\mathbb{G}_1$ together with a valid random pair (g_1^t, g_2^t).
  - If $(g_s, g_s^r, \mathsf{vk}_1, Y)$ is a Diffie-Hellman tuple, one knows that $Y = \mathsf{vk}_1^r$. $\mathcal{B}$ computes $H = -2jq_s + y_0 + \sum y_i M_i$ and $J = z_0 + \sum z_i M_i$, $\mathcal{F}(M) = h_s^H g_s^J$. If $H \equiv 0 \mod p$, it aborts, else it sets
  $$\sigma = (\mathsf{vk}_1^{-J/H} Y^{-1/H} (\mathcal{F}(M) c_0^{1/\gamma})^s, (\mathsf{vk}_1^{-1/H} g_1^s, \mathsf{vk}_2^{-1/H} g_2^s)),$$
  for some random scalar s. Setting $t = s - x/H$, we can see this is indeed a valid signature (as output as the end of the signing interactive protocol), since we have:
  $$\begin{aligned}
  \sigma_1 &= \mathsf{vk}_1^{-J/H} Y^{-1/H} (\mathcal{F}(M) c_0^{1/\gamma})^s = \mathsf{vk}_1^{-J/H} g_s^{-xr/H} (h_s^H g_s^J g_s^r)^s \\
  &= g_s^{-xJ/H} g_s^{-xr/H} (h_s^H g_s^J g_s^r)^t (h_s^H g_s^J g_s^r)^{x/H} = h^x (h^H g_s^J g_s^r)^t \\
  &= \mathsf{sk} \cdot \delta^t \qquad \text{where } \delta = \mathcal{F}(M) \times g_s^r \\
  \sigma_{2,1} &= \mathsf{vk}_1^{-1/H} g_1^s = g_1^{-x/H} g_1^s = g_1^t \\
  \sigma_{2,2} &= \mathsf{vk}_2^{-1/H} g_2^s = g_2^{-x/H} g_2^s = g_2^t
  \end{aligned}$$
  - $\mathcal{B}$ then acts honestly to send the signature through the SPHF.

  After a q_s queries, $\mathcal{A}$ outputs a valid signature σ^* on a new message M^* with non negligible probability.
- As before $\mathcal{B}$ computes $H^* = -2jq_s + y_0 + \sum y_i M_i^*$ and $J^* = z_0 + \sum z_i M_i^*$, $\mathcal{F}(M) = h^{H^*} g_1^{J^*}$.
- If $H^* \not\equiv 0 \mod p$, $\mathcal{B}$ aborts. Otherwise $\sigma^* = (\mathsf{sk} \cdot \mathcal{F}(M^*)^t, g_s^t, g_2^t) = (\mathsf{sk} \cdot g_s^{tJ^*}, g_s^t, g_2^t)$ and so $\sigma_1^*/\sigma_2^{*J^*} = \mathsf{sk} = h_s^x$. Therefore if $\mathcal{A}$'s signature is valid and if $H^* \not\equiv 0 \mod p$, $\mathcal{B}$ solves its CDH^+ challenge.

The probability that all the $H \not\equiv 0 \mod p$ for all the simulations, but $H^* \equiv 0 \mod p$ in the forgery is the $(1, q_s)$-programmability of the Waters function. A full proof showing that it happens with probability in $\Theta(\mathsf{Succ}^{\mathsf{CDH}}_{p,\mathbb{G}_1,g_1,\mathbb{G}_2,g_2}(\mathfrak{K})/q_s\sqrt{\ell})$ can be found in [HK08]. $\square$

D Concrete Constructions of SPHF

In this appendix, we extend the notations from the Section 4, and namely to explicitly handle bilinear groups. We then illustrate the formalism with more expressive languages on pairing-product equations. We focus on KV-SPHF constructions, and thus omit the possible relation Γ can have with the word $\boldsymbol{u}$.

D.1 Graded Rings for Bilinear Groups

Definition. Let us consider three multiplicative cyclic groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p. Let g_1 and g_2 be two generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ or $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ is called a *bilinear setting* if $e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$ is a bilinear map (called a pairing) with the following properties:

- *Bilinearity.* For all $(a, b) \in \mathbb{Z}_p^2$, we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$;
- *Non-degeneracy.* The element $e(g_1, g_2)$ generates $\mathbb{G}_T$;
- *Efficient computability.* The function e is efficiently computable.

It is called a *symmetric* bilinear setting if $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$. In this case, we denote it $(p, \mathbb{G}, \mathbb{G}_T, e)$ and we suppose $g = g_1 = g_2$. Otherwise, if $\mathbb{G}_1 \neq \mathbb{G}_2$, it is called an *asymmetric* bilinear setting one otherwise.

Graded Rings for Bilinear Groups. We show that graded rings can be used to deal with such bilinear groups.

Symmetric Bilinear Group. Let $(p, \mathbb{G}, \mathbb{G}_T, e)$ be a symmetric bilinear group, and g be a generator of $\mathbb{G}$. We can represent this bilinear group by a graded ring $\mathfrak{G}$ with $\kappa = 2$ and $\tau = 1$. More precisely, we can consider the following map: $[0, x]$ corresponds to $x \in \mathbb{Z}_p$, $[1, x]$ corresponds to $g^x \in \mathbb{G}$ and $[2, x]$ corresponds to $e(g, g)^x \in \mathbb{G}_T$.

Asymmetric Bilinear Group. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be an asymmetric bilinear group, and g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We can represent this bilinear group by a graded ring $\mathfrak{G}$ with $\kappa = 1$ and $\tau = 2$. More precisely, we can consider the following map: $[(0, 0), x]$ corresponds to $x \in \mathbb{Z}_p$, $[(1, 0), x]$ corresponds to $g_1^x \in \mathbb{G}_1$, $[(0, 1), x]$ corresponds to $g_2^x \in \mathbb{G}_2$ and $[(1, 1), x]$ corresponds to $e(g_1, g_2)^x \in \mathbb{G}_T$.

Notations. We have chosen an additive notation for the group law in $\mathfrak{G}_{\boldsymbol{v}}$. On the one hand, this a lot easier to write generic things done, but, on the other hand, it is a bit cumbersome for bilinear groups to use additive notations. Therefore, when we provide an example with a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, we use multiplicative notation $\cdot$ for the law in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, and additive notation $+$ for the law in $\mathbb{Z}_p$, as soon as it is not too complicated. But when needed, we will also use the notation $\oplus$ and $\odot$ which correspond to the addition law and the multiplicative law of the corresponding graded rings. In other words, for any $x, y \in \mathbb{Z}_p$, $u_1, v_1 \in \mathbb{G}_1$, $u_2, v_2 \in \mathbb{G}_2$ and $u_T, v_T \in \mathbb{G}_T$, we have:

$$\begin{aligned}
x \oplus y &= x + y & & & x \odot y &= x \cdot y = xy \\
u_1 \oplus v_1 &= u_1 \cdot v_1 = u_1 v_1 & & & x \odot u_1 &= u_1^x \\
u_2 \oplus v_2 &= u_2 \cdot v_2 = u_2 v_2 & & & x \odot u_1 &= u_1^x \\
u_T \oplus v_T &= u_T \cdot v_T & u_1 \odot u_2 &= e(u_1, u_2) & x \odot u_T &= u_T^x.
\end{aligned}$$

D.2 SPHF for Linear Pairing Equations over Ciphertexts

We first construct an SPHF for a linear pairing equation in an asymmetric bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$ over ElGamal commitments. This will actually be a KV-SPHF, and a particular case of the construction of the next section for quadratic pairing equation. It is thus a warm-up for this technical section. The construction can obviously be extended to systems of linear pairing equations, and to other commitments schemes using the same methods as in Section 5. It can also be slightly simplified in the case of symmetric bilinear groups.

Notations. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be a (asymmetric) bilinear group. Let g_1, g_2 be generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively, and let $g_T = e(g_1, g_2)$. Let $\mathsf{ek}_1 = (g_1, h_1 = g_1^{x_1})$, $\mathsf{ek}_2 = (g_2, h_2 = g_2^{x_2})$ and $\mathsf{ek}_T = (g_T, h_T = g_T^{x_T})$ be ElGamal key for encryption scheme in, respectively, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$.

We are interested in languages of commitments $(C_{1,i})_i$ of $(X_{1,i})_i \in \mathbb{G}_1^{n_1}$, $(C_{2,j})_j$ of $(X_{2,j})_j \in \mathbb{G}_2^{n_2}$, and $(C_{T,k})_i$ of $(X_{T,k})_k \in \mathbb{G}_T^{n_T}$ such that:

$$\prod\nolimits_i e(X_{1,i}, A_{2,i}) \cdot \prod\nolimits_j e(A_{1,j}, X_{2,j}) \cdot \prod\nolimits_k X_{T,k}^{a_{T,k}} = B, \tag{2}$$

with $\mathsf{aux} = ((A_{1,j})_j, (A_{2,i})_i, (a_{T,k})_k) \in \mathbb{G}_1^{n_2} \times \mathbb{G}_2^{n_1} \times \mathbb{Z}_p^{n_T}$. This can also be written:

$$\left(\bigoplus_{i=1}^{n_1} A_{2,i} \odot X_{1,i}\right) \oplus \left(\bigoplus_{j=1}^{n_2} A_{1,j} \odot X_{2,j}\right) \oplus \left(\bigoplus_{k=1}^{n_T} a_{T,k} \odot X_{T,k}\right) = B.$$

Let us also write, for any $\omega \in \{1, 2, T\}$ and $\iota \in \{1, \ldots, n_\omega\}$: $C_{\omega,\iota} = (u_{\omega,\iota} = g_\omega^{r_{\omega,\iota}}, e_{\omega,\iota} = h_\omega^{r_{\omega,\iota}} X_{\omega,\iota})$. Words of $\mathscr{S}et$ are tuple $C = \boldsymbol{u} = (C_{\omega,\iota})_{\omega \in \{1,2,T\},\, \iota \in \{1,\ldots,n_\omega\}}$.

Basic Scheme in $\mathbb{G}_T$. Let us consider

$$\Gamma = \begin{pmatrix} g_1 & 0 & 0 & h_1 \\ 0 & g_2 & 0 & h_2 \\ 0 & 0 & g_T & h_T \end{pmatrix} \qquad \Theta(C) = \begin{pmatrix} \bigoplus_i A_{2,i} \odot u_{1,i}, \bigoplus_j A_{1,j} \odot u_{2,j}, \bigoplus_k a_{T,k} \odot u_{T,k}, \\ (\bigoplus_i A_{2,i} \odot e_{1,i}) \oplus \left(\bigoplus_j A_{1,j} \odot e_{2,j}\right) \oplus (\bigoplus_k a_{T,k} \odot e_{T,k}) \ominus B \end{pmatrix}.$$

Because of the diagonal block in Γ, one can note that the unique possibility is

$$\boldsymbol{\lambda} = (\bigoplus_i A_{2,i} \odot r_{1,i}, \bigoplus_j A_{1,j} \odot r_{2,j}, \bigoplus_k r_{T,k}) = (\prod_i A_{2,i}^{r_{1,i}}, \prod_j A_{1,j}^{r_{2,j}}, \sum_k r_{T,k}).$$

We then have $\boldsymbol{\lambda} \odot \Gamma = \Theta(C)$ if and only if

$$\prod\nolimits_i e(h_1^{r_{1,i}}, A_{2,i}) \cdot \prod\nolimits_j e(A_{1,j}, h_2^{r_{2,j}}) \cdot \prod\nolimits_k h_T^{r_{T,k}} = \prod\nolimits_i e(e_{1,i}, A_{2,i}) \cdot \prod\nolimits_j e(A_{1,j}, e_{2,j}) \cdot \prod\nolimits_k e_{T,k}^{a_{T,k}} / B$$

and thus if and only if Equation (2) is true, *i.e.*, the word is in the language. Furthermore, if we set $\gamma_1 = g_1^{\alpha_1} h_1^{\alpha^4}$, $\gamma_2 = g_2^{\alpha_2} h_2^{\alpha^4}$, and $\gamma_3 = g_T^{\alpha_3} h_T^{\alpha^4}$, we have

$$\begin{aligned} H &= \left(\prod_{i=1}^{n_1} e(u_{1,i}, A_{2,i})\right)^{\alpha_1} \cdot \left(\prod_{j=1}^{n_2} e(A_{1,j}, u_{2,j})\right)^{\alpha_2} \cdot \left(\prod_{k=1}^{n_T} u_{T,k}^{a_{T,k}}\right)^{\alpha_3} \\ &\times \left(\prod_{i=1}^{n_1} e(e_{1,i}, A_{2,i}) \cdot \prod_{j=1}^{n_2} e(A_{1,j}, e_{2,j}) \cdot \prod_{k=1}^{n_T} e_{T,k}^{a_{T,k}} / B\right)^{\alpha_4} \\ &= e(\gamma_1, \prod\nolimits_i A_{2,i}^{r_{1,i}}) \cdot e(\prod\nolimits_j A_{1,j}^{r_{2,j}}, \gamma_2) \cdot \gamma_3^{\sum_k r_{T,k}} && = H'. \end{aligned}$$

Variant. The above scheme is not efficient enough for practical use because elements in $\mathbb{G}_T$ are often big and operations in $\mathbb{G}_T$ are often slow. If $h_T = e(h_1, g_2)$, then the last row of Γ can be $(0, 0, g_1, h_1)$ which enables faster hashing and shorter projection key. We remark this modified encryption scheme in $\mathbb{G}_T$ is $\mathsf{IND\text{-}CPA}$ as soon as DDH is hard in $\mathbb{G}_1$, which we need to suppose for the ElGamal encryption scheme in $\mathbb{G}_1$ to be $\mathsf{IND\text{-}CPA}$. So this variant is always more efficient when using ElGamal encryption.

However, if DDH is easy, as in symmetric bilinear group, this variant may not be interesting, since it requires to use the linear encryption scheme in $\mathbb{G}_T$ instead of the ElGamal one.

D.3 SPHF for Quadratic Pairing Equations over Ciphertexts

In this section, we present a $\mathsf{KV\text{-}SPHF}$ for language of ElGamal commitments verifying a quadratic pairing equation. As usual, it can be extended to systems of quadratic pairing equations, and to other commitments schemes. We use the same notations as in Section D.2.

Example. Before showing the generic construction, we describe it on a simple example: we are interested in languages of the ciphertexts $C_1 = (u_1 = g_1^{r_1}, e_1 = h_1^{r_1} X_1)$ and $C_2 = (u_2 = g_2^{r_2}, e_2 = h_2^{r_2} X_2)$, that encrypt two values X_1 and X_2 such that $e(X_1, X_2) = B$ where B is some constant in $\mathbb{G}_T$ and $\mathsf{aux} = B$. We remark the equation $e(X_1, X_2) = B$ can also be written $X_1 \odot X_2 = B$. Let us consider

$$\Gamma = \begin{pmatrix} g_1 \odot g_2 & 0 & 0 & h_1 \odot h_2 \\ 0 & g_1 & 0 & h_1 \\ 0 & 0 & g_2 & h_2 \end{pmatrix} \qquad \begin{aligned} \Theta(C) &= (-u_1 \odot u_2, u_1 \odot e_2, e_1 \odot u_2, e_1 \odot e_2 \ominus B) \\ &= (e(u_1, u_2)^{-1}, e(u_1, e_2), e(e_1, u_2), e(e_1, e_2)/B). \end{aligned}$$

Because of the diagonal block in Γ, one can note that the unique possibility is

$$\boldsymbol{\lambda} = (-r_1 r_2, r_1 \odot e_2, r_2 \odot e_1) = (-r_1 r_2, e_2^{r_1}, e_1^{r_2}).$$

We have $\boldsymbol{\lambda} \odot \Gamma = \Theta(C)$ if and only if $e(h_1, h_2)^{-r_1 r_2} \cdot e(h_1, e_2^{r_1}) \cdot e(e_1^{r_2}, h_2) = e(e_1, e_2)/B$, and thus,

$$\begin{aligned} B &= e(e_1, e_2)/(e(h_1^{r_1}, X_2) \cdot e(e_1, h_2^{r_2})) \\ &= e(e_1, X_2)/e(h_1^{r_1}, X_2) = e(X_1, X_2) \end{aligned}$$

For the sake of completeness, if $\gamma_1 = e(g_1, g_2)^{\alpha_1} e(h_1, h_2)^{\alpha_4}$, $\gamma_2 = g_1^{\alpha_2} h_1^{\alpha_4}$, and $\gamma_3 = g_2^{\alpha_3} h_2^{\alpha_4}$, the corresponding hash value is:

$$H = e(u_1, u_2)^{-\alpha_1} \cdot e(u_1, e_2)^{\alpha_2} \cdot e(e_1, u_2)^{\alpha_3} \cdot (e(e_1, e_2)/B)^{\alpha_4} = \gamma_1^{-r_1 r_2} \cdot e(\gamma_2, e_2^{r_1}) \cdot e(e_1^{r_2}, \gamma_3).$$

Notations. Let us now introduce notation to handle any quadratic equation. In addition to previous notations, as in Section D.2, we also write $\mathsf{ek}_T = (g_T, h_T = g_T^{x_T})$ a public key for ElGamal encryption scheme in $\mathbb{G}_T$. We are interested in languages of commitments $(C_{1,i})_i$ of $(X_{1,i})_i \in \mathbb{G}_1^{n_1}$, $(C_{2,j})_j$ of $(X_{2,j})_j \in \mathbb{G}_2^{n_2}$, and $(C_{T,k})_i$ of $(X_{T,k})_k \in \mathbb{G}_T^{n_T}$ such that:

$$\prod\nolimits_i e(X_{1,i}, A_{2,i}) \cdot \prod\nolimits_j e(A_{1,j}, X_{2,j}) \cdot \prod\nolimits_i \prod\nolimits_j e(X_{1,i}, X_{2,j})^{a_{i,j}} \cdot \prod\nolimits_k X_{T,k}^{a_{T,k}} = B, \tag{3}$$

with $\mathsf{aux} = ((A_{2,i})_i, (A_{1,j})_j, (a_{i,j})_{i,j}, (a_{T,k})_k) \in \mathbb{G}_1^{n_1} \times \mathbb{G}_2^{n_2} \times \mathbb{Z}_p^{n_1 n_2 + n_T}$. This can also be written:

$$\left(\bigoplus_{i=1}^{n_1} A_{2,i} \odot X_{1,i}\right) \oplus \left(\bigoplus_{j=1}^{n_2} A_{1,j} \odot X_{2,j}\right) \oplus \left(\bigoplus_{i=1}^{n_1} \bigoplus_{j=1}^{n_2} a_{i,j} \odot X_{1,i} \odot X_{2,j}\right) \oplus \left(\bigoplus_{k=1}^{n_T} a_{T,k} \odot X_{T,k}\right) = B.$$

Let us also write, for any $\omega \in \{1, 2, T\}$ and $\iota \in \{1, \dots, n_\omega\}$: $C_{\omega,\iota} = (u_{\omega,\iota} = g_\omega^{r_{\omega,\iota}}, e_{\omega,\iota} = h_\omega^{r_{\omega,\iota}} X_{\omega,\iota})$.

Basic Scheme in $\mathbb{G}_T$. Let us consider the following matrix, with a diagonal block

$$\Gamma = \begin{pmatrix} g_1 \odot g_2 & 0 & 0 & 0 & h_1 \odot h_2 \\ 0 & g_1 & 0 & 0 & h_1 \\ 0 & 0 & g_2 & 0 & h_2 \\ 0 & 0 & 0 & g_T & h_T \end{pmatrix}$$

With

$$\Theta(C) = \begin{pmatrix} \bigoplus_i \bigoplus_j -a_{i,j} \odot u_{1,i} \odot u_{2,j}, \left(\bigoplus_i \bigoplus_j a_{i,j} \odot u_{1,i} \odot e_{2,j}\right) \oplus (\bigoplus_i A_{2,i} \odot u_{1,i}), \\ \left(\bigoplus_i \bigoplus_j a_{i,j} \odot e_{1,i} \odot u_{2,j}\right) \oplus \left(\bigoplus_j A_{1,j} \odot u_{2,j}\right), \bigoplus_i a_{T,i} \odot u_{T,i}, \\ \left(\bigoplus_i \bigoplus_j a_{i,j} \odot e_{2,i} \odot e_{2,j}\right) \oplus (\bigoplus_i A_{2,i} \odot e_{1,i}) \oplus \left(\bigoplus_j A_{1,j} \odot e_{2,j}\right) \oplus (\bigoplus_k a_{T,k} \odot e_{T,k}) \ominus B \end{pmatrix}$$

the requirement $\boldsymbol{\lambda} \odot \Gamma = \Theta(C)$ implies

$$\begin{aligned} \boldsymbol{\lambda} &= \begin{pmatrix} \bigoplus_i \bigoplus_j -a_{i,j} \odot r_{1,i} \odot r_{2,j}, \left(\bigoplus_i \bigoplus_j r_{1,i} \odot a_{i,j} \odot e_{2,j}\right) \oplus (\bigoplus_i A_{2,i} \odot r_{1,i}), \\ \left(\bigoplus_i \bigoplus_j r_{2,i} \odot a_{i,j} \odot e_{1,j}\right) \oplus \left(\bigoplus_j A_{1,j} \odot r_{2,j}\right), \bigoplus_k r_{T,k} \end{pmatrix} \\ &= \left(\sum_i \sum_j a_{i,j} r_{1,i} r_{2,j}, \prod_i \prod_j e_{2,j}^{r_{1,i} a_{i,j}} \cdot \prod_i A_{2,i}^{r_{1,i}}, \prod_i \prod_j e_{1,j}^{r_{2,i} a_{i,j}} \cdot \prod_j A_{1,j}^{r_{2,j}}, \sum_k r_{T,k} \right), \end{aligned}$$

and it is satisfied, if and only if Equation (3) is true, *i.e.*, the word is in the language.

Variant. The same trick as the one used in the variant of the SPHF of Section D.2 can be used to avoid having too many elements of the projection key in $\mathbb{G}_T$.