Disjunctions for Hash Proof Systems: New Constructions and Applications

Michel Abdalla, Fabrice Benhamouda, and David Pointcheval

ENS, Paris, France *
{michel.abdalla, fabrice.ben.hamouda, david.pointcheval}@ens.fr

Abstract. Smooth Projective Hash Functions (SPHFs), also known as Hash Proof Systems, were first introduced by Cramer and Shoup (Eurocrypt'02) as a tool to construct efficient IND-CCA secure encryption schemes. Since then, SPHFs have been used in various applications, including password authenticated key exchange, oblivious transfer, and zero-knowledge arguments. What makes SPHFs so interesting and powerful is that they can be seen as implicit proofs of membership for certain languages. As a result, by extending the family of languages that they can handle, one often obtains new applications or new ways to understand existent schemes. In this paper, we show how to construct SPHFs for the disjunction of languages defined generically over cyclic, bilinear, and multilinear groups. Among other applications, this enables us to construct the most efficient one-time simulation-sound (quasi-adaptive) non-interactive zero-knowledge arguments for linear languages over cyclic groups, and the first one-round group password-authenticated key exchange without random oracles.

Keywords. Smooth Projective Hash Function, Non-Interactive Zero-Knowledge Proof, Group Password Authenticated Key Exchange, Threshold Encryption Scheme.

1 Introduction

Smooth Projective Hash Functions (SPHFs) and Applications. Hash Proof Systems or Smooth Projective Hash Functions (SPHFs) were first introduced by Cramer and Shoup [CS02] in order to achieve IND-CCA security from IND-CPA encryption schemes, which led to the first efficient IND-CCA encryption scheme provably secure in the standard model under the DDH assumption [CS98]. They can be seen as a kind of implicit designated-verifier proofs of membership [ACP09, BPV12]. Informally speaking, SPHFs are families of pairs of functions $(\mathsf{Hash}, \mathsf{ProjHash})$ defined on a language $\mathscr{L} \subset \mathcal{X}$. These functions are indexed by a pair of associated keys $(\mathsf{hk}, \mathsf{hp})$, where the hashing key hk and the projection key hp can be seen as the private and public keys, respectively. When computed on a word $C \in \mathscr{L}$, both functions should lead to the same result: $\mathsf{Hash}(\mathsf{hk}, \mathscr{L}, C)$ with the hashing key and $\mathsf{ProjHash}(\mathsf{hp}, \mathscr{L}, C, w)$ with the projection key only but also a witness w that $C \in \mathscr{L}$. Of course, if $C \notin \mathscr{L}$, such a witness does not exist, and the smoothness property states that $\mathsf{Hash}(\mathsf{hk}, \mathscr{L}, C)$ is independent of hp. As a consequence, even knowing hp, we cannot guess $\mathsf{Hash}(\mathsf{hk}, \mathscr{L}, C)$.

Since its introduction, SPHFs have been used in various applications, including Password Authenticated Key Exchange (PAKE) [KOY01, GL03, KV11], Oblivious Transfer [Kal05, ABB+13], One-Time Relatively-Sound Non-Interactive Zero-Knowledge Arguments [JR12], Zero-Knowledge Arguments [BBC+13], and Trapdoor SPHFs (TSPHFs) [BBC+13].

Password Authenticated Key Exchange (PAKE). The first main application (after the construction of IND-CCA encryption schemes) is certainly the construction of PAKE schemes in the standard model. Such a PAKE scheme enables two users to agree on a common session key while using a simple password (a low-entropy secret) as authentication means. For such protocols, online dictionary attacks, which consist in guessing the password of an honest user and running honestly the protocol with this guessed password, are unavoidable. But their impact can be

* CNRS – UMR 8548 and INRIA – EPI Cascade

mitigated by organizational means. That is why a PAKE scheme is considered secure if an adversary can only check one guessed password per interaction with an honest party. Unfortunately, this makes the design of PAKE significantly more complex than the design of authenticated key exchange protocols in settings where users hold public/private key pairs.

Gennaro and Lindell [GL03] proposed a generic construction of PAKE in the security model of Bellare, Pointcheval and Rogaway [BPR00] (BPR). This is a generalization of the Katz-Ostrovsky-Yung protocol [KOY01]. The Gennaro-Lindell framework for PAKE basically consists, for each player, in sending a commitment of the password, and a projection key to check the validity of the partner's commitment. More recently, Katz and Vaikuntanathan [KV11] proposed a stronger definition for SPHF enabling the design of a one-round PAKE protocol, still based on the Gennaro-Lindell framework, where the two players only send one flow and both flows can be sent simultaneously.

An interesting generalization of PAKE are Group PAKE or GPAKE protocols, which enable more than two users sharing a common passwords to agree on a common session key. So far, the only known GPAKE in the standard model are at least two-round [ABCP06].

Zero-Knowledge Arguments and Trapdoor SPHFs (TSPHFs). A second important application of SPHFs is in the construction of zero-knowledge arguments. Intuitively, being able to compute the hash value of a word C, when only given hp, implicitly proves that $C \in \mathscr{L}$. That is exactly what is used for the construction of IND-CCA encryption schemes and PAKE.

Therefore, an SPHF directly yields a two-round honest-verifier zero-knowledge argument for $\mathscr{L}$: the verifier first generates a hashing key hk and an associated projection key hp, sends the projection key hp to the prover who answers with the hash value H of C under hp, which he can compute using his witness w of $C \in \mathscr{L}$. Then the verifier accepts if and only if H is equal to the hash value of C he can compute using hk. On the one hand, this protocol is honest-verifier zero-knowledge, since if the verifier behaves correctly and $C \in \mathscr{L}$, then he already knows the answer of the prover, because he can compute it himself with hk. On the other hand, this protocol is statistically sound, thanks to the smoothness of the SPHF which ensures that, if $C \notin \mathscr{L}$, then its hash value appears random and is thus unpredictable given only hp.

In the case of malicious verifiers, Benhamouda et al. in [BBC+13] remarked that the protocol above is not necessarily zero-knowledge. To address this problem, they proposed a variant of SPHF, called Trapdoor SPHF or TSPHF, which provides a two-round zero-knowledge protocol in the common reference string model. In the new variant, the common reference string has an associated trapdoor which allows the computation of the hash value of any word $C \in \mathcal{X}$ when given a projection key hp. Since hp now needs to contain enough information to compute the hash value of any word in $\mathcal{X}$, the smoothness property of TSPHFs is no longer statistical but computational. As for the zero-knowledge property, as shown in [BBC+13], it follows from the fact that one can use the new trapdoor to simulate a proof against any verifier.

It is worth remarking that the two constructions of zero-knowledge arguments mentioned above are interactive as they require two rounds. Until now, it was not known how to build NIZK (non-interactive zero-knowledge arguments) from SPHFs or TSPHFs.

Witness Encryption. One of the most recent applications of SPHFs is the notion of witness encryption introduced by Garg et al. in [GGSW13]. In a witness encryption scheme, a user can encrypt a message to a word/statement C and a language $\mathscr{L}$, and any user knowing a witness w for $C \in \mathscr{L}$ can decrypt the ciphertext c. Such a scheme is secure if no polynomial-time adversary can distinguish between the encryption of two (equal-length) plaintexts, when $C \notin \mathscr{L}$.

An SPHF for a language $\mathscr{L}$ directly yields such a scheme for $\mathscr{L}$: to encrypt a message M for a word C, we pick a random hashing key hk for $\mathscr{L}$, then derive the associated projection key hp, and the ciphertext consists of the pair $c = (\mathsf{hp}, H \mathbin{\mathsf{xor}} M)$, where H is the hash value of C under hk. Correctness follows from the fact that, when $C \in \mathscr{L}$, any user knowing a witness w for $C \in \mathscr{L}$, can compute H using hp and w. On the other hand, when $C \notin \mathscr{L}$, the smoothness property states that H looks completely random given only hp. Hence the resulting scheme is statistically secure.

Before continuing with the introduction, we need to highlight the fact there are various definitions of SPHFs, as explained in [BBC+13]: the projection key hp may or may not depend on C, and if hp is independent of C, the smoothness may or may not hold even if C is chosen after having seen hp. For witness encryption, for example, the weakest notion (hp depends on C) is sufficient, while for encryption schemes or one-round PAKE, the strongest notion (hp does not depend on C and C may be chosen after hp in smoothness) is required. In this article, we focus on the strongest notion of SPHF, also called KV-SPHF in [BBC+13], since it has more applications. However, most parts of the paper could be adapted to the weakest notion.

Expressiveness of SPHFs. Due to the wide range of applications of SPHFs, one may wonder what kind of languages can be handled by SPHFs.

First, since SPHF implies statistical witness encryption, it is important to remark that it is impossible to construct SPHF for any NP language, unless the polynomial hierarchy collapses [GGSW13]. Nevertheless, as the many different applications show, the class of languages supported by SPHFs can be very rich.

Diverse Groups and Diverse Vector Spaces. In [CS02], Cramer and Shoup showed that SPHFs can handle any language based on what they call a diverse group. Most, if not all, constructions of SPHF are based on diverse groups. However, in the context of languages over cyclic groups, bilinear groups or even multi-linear groups, diverse groups may appear slightly too generic. That is why Benhamouda et al. introduced a generic framework (later called *diverse vector space*) encompassing most, if not all, known SPHFs based over this kind of groups. It can be seen as particular diverse groups with more mathematical structure, namely using vector spaces instead of groups. In this article, we are mainly interested on SPHFs based on diverse vector spaces.

Operations on SPHFs. We may then wonder which "operations" on languages can be applied to SPHFs.

In [ACP09], Abdalla, Chevalier and Pointcheval showed that, if we have SPHFs for languages $\mathscr{L}_1$ and $\mathscr{L}_2$, then we can construct an SPHF for the *conjunction* and the *disjunction* of $\mathscr{L}_1$ and $\mathscr{L}_2$.

The conjunction of $\mathscr{L}_1$ and $\mathscr{L}_2$ is the language $\mathscr{L} = \mathscr{L}_1 \times \mathscr{L}_2$, so that a word $C = (C_1, C_2) \in \mathscr{L}$ if and only if $C_1 \in \mathscr{L}_1$ and $C_2 \in \mathscr{L}_2$. This conjunction is a generalization of the "classical" conjunction: $C_1 \in \mathscr{L}$ if and only if $C_1 \in \mathscr{L}_1$ and $C_1 \in \mathscr{L}_2$, which we can get by setting $C_1 = C_2$.

The disjunction of $\mathscr{L}_1$ and $\mathscr{L}_2$ is defined similarly as the language $\mathscr{L} = (\mathscr{L}_1 \times \mathcal{X}_2) \cup (\mathcal{X}_1 \times \mathscr{L}_2)$, so that $C = (C_1, C_2) \in \mathscr{L}$ if and only if $C_1 \in \mathscr{L}_1$ or $C_2 \in \mathscr{L}_2$. An important property for the disjunction is that a witness for $C = (C_1, C_2) \in \mathscr{L}$ can be either a witness w_1 for $C_1 \in \mathscr{L}_1$ or a witness w_2 for $C_2 \in \mathscr{L}_2$. As previously, if we set $C_1 = C_2$, we get the "classical" disjunction: $C = (C_1, C_1) \in \mathscr{L}$ if and only if $C_1 \in \mathscr{L}_1$ or $C_1 \in \mathscr{L}_2$.

Unfortunately, while the conjunction construction in [ACP09] yields the strongest form of SPHF if the two original SPHFs were of this form, this is not the case for the disjunction construction, where the projection key hp necessarily depends on C. And this greatly limits its applications.

A reader familiar to [Gro06] may wonder why, since SPHFs exist for languages of quadratic pairing equations over commitments [BBC+13], methods in [Gro06] cannot be applied to provide a form of disjunction. But unfortunately, this would not yield a real SPHF, since additional commitments would be required.

1.1 Results

Disjunction of SPHFs. The first main result of our article is to show how to construct the disjunction of two SPHFs for two languages based on diverse vector spaces. Essentially, the only requirement for the construction is that it is possible to compute a pairing between an element of the first language $\mathscr{L}_1$ and an element of the second language $\mathscr{L}_2$. Concretely, if we have a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of some prime order p (we say that $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ is a bilinear group), and if $\mathscr{L}_1$ is defined over $\mathbb{G}_1$ and $\mathscr{L}_2$ over $\mathbb{G}_2$, then our construction provides an SPHF for the disjunction of $\mathscr{L}_1$ and $\mathscr{L}_2$.

This disjunction can be repeated multiple times, if multi-linear maps are available. The only limitation is that the efficiency of our constructions grows exponentially with the number of repetitions, therefore limiting the total number of disjunctions that we can compute.

Applications. In the following, we present some interesting applications of our new construction for disjunctions.

Constant-Size NIZK and One-Time Simulation-Sound NIZK. First, we show how to use disjunctions of SPHFs to create efficient *NIZK* and even *one-time simulation-sound NIZK* , i.e., NIZK in which a dishonest (polynomial-time) prover cannot produce a valid proof of a false statement, even if he saw one simulated proof on a statement of its choice (which may be false). The proof size consists of only two group elements, even for the one-time simulation-sound version, assuming the language we are interested in can be handled by an SPHF over some group $\mathbb{G}_1$, where $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is an asymmetric bilinear group, and assuming DDH is hard in $\mathbb{G}_2$. The languages handled roughly consist in languages which can be defined by "linear" equations over $\mathbb{G}_1$, such as the DDH language, the language of valid Cramer-Shoup [CS98] ciphertexts and many other useful languages as shown in [BBC$^+$13, JR13].

Our NIZK is slightly different from a usual NIZK, since the common reference string depends on the language. Jutla and Roy called them quasi-adaptive NIZK in [JR13], and showed that they can replace NIZK in most applications.

Our one-time simulation-sound NIZK yields a very efficient structure-preserving threshold IND-CCA encryption scheme, with the shortest ciphertext size so far. Threshold means the decryption key can be shared between parties and a ciphertext can be decrypted if and only if enough parties provide a partial decryption of it using their key share, while structure-preserving means it can be used in particular with Groth-Sahai NIZK [GS08] or our new NIZK!

Other Applications. Another important application is the first *one-round GPAKE* with n players, assuming the existence of $(n-1)$-way multi-linear maps and the hardness of the n-linear assumption n-Lin [CLT13][1] without random oracles. This was an open problem. We remark, however, that are our construction only works for small values of n since the the overall complexity of the protocol and the gap in the security reduction grow exponentially in n. Anyway, the tri-partite PAKE that just makes use of pairings is quite efficient since it consists of flows with 61 group elements only for each users (5 for the Cramer-Shoup ciphertext and 56 for the projection key).

A final application is a new construction for TSPHF, which supports slightly more languages than the one in [BBC$^+$13], but which is slightly less efficient.

Pseudo-Randomness Projective Hash Functions (PrPHFs) and More Efficient Applications. For our NIZK and our new TSPHF, the construction essentially consists in the

[1] In the original construction of Garg, Gentry and Halevi [GGH13], n-Lin is trivial, but it seems not to be the case in the one of Coron, Lepoint and Tibouchi [CLT13]. In any case, for $n = 3$ users, we can just use a symmetric bilinear group over elliptic curves.

disjunction of an SPHF for the language in which we are interested, and another SPHF for a language which is used to provide extra features (zero-knowledge and "public verifiability" for our NIZK and trapdoor for our TSPHF). This second language $\mathscr{L}_2$ is supposed to be a hard subset membership one, i.e., it is hard to distinguish a random word $C_2 \in \mathscr{L}_2$ from a random word $C_2 \in \mathcal{X}_2 \setminus \mathscr{L}_2$.

To get more efficient applications, we introduce the notion of pseudo-random projective hash functions (PrPHFs) which are particular SPHFs over trivial languages, i.e., languages $\mathscr{L} = \mathcal{X}$, where all words are in the language. Of course, smoothness becomes trivial, in this case. That is why PrPHFs are supposed to have another property called *pseudo-randomness*, which ensures that if the parameters of the language $\mathscr{L}$ and the word C are chosen at random, given a projection key hp (and no witness for C), the hash value H of C appears random.

We then show that we can replace the second hard subset membership language in our NIZK and our TSPHF by a trivial language with a PrPHF, assuming a certain property over the first language $\mathscr{L}_1$ (which is almost always verified). This conversion yields slightly shorter proofs (for our NIZK and our one-time simulation-sound NIZK) or slightly shorter projection keys (for our TSPHF).

Related Work. Until now, the most efficient NIZK for similar languages was the one of Jutla and Roy [JR14], and the most efficient one-time simulation-sound NIZK was actually an unbounded one-time simulation-sound NIZK from Libert et al. [LPJY14]. Both were constant-size too, but our second NIZK is slightly more efficient for κ-linear assumptions, with $\kappa \geq 2$, while our one-time simulation-sound NIZK are about ten times shorter. In addition, in our opinion, our construction seems to be more modular and simpler to understand. We provide a detailed comparison in Section 8.3.

1.2 Organization

In the following section, we give the high level intuition for all our constructions. Then, after recalling some preliminaries in Section 3, we give the details of our construction of disjunctions of SPHFs in Section 4, which is one of the main contributions. We then show how to build efficient NIZK and one-time simulation-sound NIZK from it in Section 5. Our two other applications, namely one-round GPAKE and TSPHF, are presented in Section 6. After that, we introduce the notion of PrPHF in Section 7 and show in Section 8 and Section 9 how this can improve our previous applications. These three last sections are much more technical: although the underlying ideas are similar to the ones in previous sections, the proofs are more complex.

2 Overview of Our Constructions

2.1 Disjunction of Languages

Intuition. From a very high point of view, the generic framework [BBC$^+$13] enables to construct an SPHF for any language $\mathscr{L}$ which is a subspace of the vector space of all words $\mathcal{X}$.

It is therefore possible to do the conjunction of two languages $\mathscr{L}_1$ and $\mathscr{L}_2$ supported by this generic framework by remarking that $\mathscr{L}_1 \times \mathscr{L}_2$ is a subspace of the vector space $\mathcal{X}_1 \times \mathcal{X}_2$. This construction of conjunctions is an "algebraic" version of the conjunction proposed in [ACP09].

Unfortunately, the same approach cannot be directly applied to the case of disjunctions, because $(\mathscr{L}_1 \times \mathcal{X}_2) \cup (\mathcal{X}_1 \times \mathscr{L}_2)$ is not a subspace of $\mathcal{X}_1 \times \mathcal{X}_2$, and the subspace generated by this space is $\mathcal{X}_1 \times \mathcal{X}_2$. In this article, we solve this issue by remarking that, instead of using $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$, we can consider the tensor product of $\mathcal{X}_1$ and $\mathcal{X}_2$: $\mathcal{X} = \mathcal{X}_1 \otimes \mathcal{X}_2$. Then the disjunction of $\mathscr{L}_1$ and $\mathscr{L}_2$ can be seen as the subspace $\mathscr{L}$ of $\mathcal{X}$ generated by: $\mathscr{L}_1 \otimes \mathcal{X}_2$ and $\mathcal{X}_1 \otimes \mathscr{L}_2$. Notice that $(\mathscr{L}_1 \otimes \mathcal{X}_2) \cup (\mathcal{X}_1 \otimes \mathscr{L}_2)$ is not a subspace and so $\mathscr{L}$ is much larger than this set. But we can prove that if $C_1 \otimes C_2 \in \mathscr{L}$, then $C_1 \in \mathscr{L}_1$ or $C_2 \in \mathscr{L}_2$.

Before providing more details about these constructions, let us first briefly recall the main ideas of the generic framework for constructing SPHFs.

Generic Framework for SPHFs. The generic framework for SPHFs in [BBC+13] uses a common formalization for cyclic groups, bilinear groups, and even multilinear groups (of prime order p), called graded rings. Basically, graded rings enable to use a ring structure over these groups: the addition and the multiplication of two elements u and v, denoted $u \oplus v$ and $u \odot v$, respectively, correspond to the addition and the multiplication of their discrete logarithms. For example, if g is a generator of a cyclic group $\mathbb{G}$, and a and b are two scalars in $\mathbb{Z}_p$, $a \oplus b = a + b$ (because the "discrete logarithm" of a scalar is the scalar itself), $g^a \oplus g^b = g^{a+b}$, and $g^a \odot b = g^{a \cdot b}$.

Of course, computing $g^a \odot g^b = g^{a \cdot b}$ requires a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, if the discrete logarithms a and b of g^a and g^b are not known. And if such bilinear map exists: $g^a \odot g^b = e(g^a, g^b)$. For the same reason, multiplying via $\odot$ three group elements would require a trilinear map. Therefore, graded rings can be seen as the ring $\mathbb{Z}_p$ with some limitations on the multiplication.

From a very high level point of view, in this framework, we suppose there exists a map θ from the set of words $\mathcal{X}$ to a vector space $\hat{\mathcal{X}}$ of dimension n, together with a subspace $\hat{\mathscr{L}}$ of $\hat{\mathcal{X}}$, generated by a family of vectors $(\boldsymbol{\Gamma_i})_{i=1,\dots,k}$, such that $C \in \mathscr{L}$ if and only if $\hat{\boldsymbol{C}} := \theta(C) \in \hat{\mathscr{L}}$. A witness for a word $C \in \mathscr{L}$ is a vector $\boldsymbol{\lambda} = (\lambda_i)_{i=1,\dots,k}$ so that

$$\hat{\boldsymbol{C}} = \theta(C) = \bigoplus_{i=1}^{k} \lambda_i \odot \boldsymbol{\Gamma_i}.$$

In other words, it consists of the coefficients of a linear combination of $(\boldsymbol{\Gamma_i})_{i=1,\dots,k}$ equal to $\hat{\boldsymbol{C}}$.

Then, a hashing key hk is just a random linear form $\mathsf{hk} := \alpha \in \mathcal{X}^*$, and the associated projection key is the vector of its values on $\boldsymbol{\Gamma_1}, \dots, \boldsymbol{\Gamma_k}$:

$$\mathsf{hp} := \boldsymbol{\gamma} = (\gamma_i)_{i=1,\dots,k} = (\alpha(\boldsymbol{\Gamma_i}))_{i=1,\dots,k}.$$

The hash value of a word C is then $H := \alpha(\hat{\boldsymbol{C}})$, and if $\boldsymbol{\lambda}$ is a witness for $C \in \mathscr{L}$, this hash value can also be computed as:

$$H = \alpha(\hat{\boldsymbol{C}}) = \alpha\left(\bigoplus_{i=1}^{k} \lambda_i \odot \boldsymbol{\Gamma_i}\right) = \bigoplus_{i=1}^{k} \lambda_i \odot \alpha(\boldsymbol{\Gamma_i}) = \bigoplus_{i=1}^{k} \lambda_i \odot \gamma_i,$$

which only depends on the witness $\boldsymbol{\lambda}$ and the projection key hp.

The smoothness comes from the fact that, if $C \notin \mathscr{L}$, then $\hat{\boldsymbol{C}} \notin \hat{\mathscr{L}}$ and $\hat{\boldsymbol{C}}$ is linearly independent from $(\boldsymbol{\Gamma_i})_{i=1,\dots,k}$. Hence, $\alpha(\hat{\boldsymbol{C}})$ looks random even given $\mathsf{hp} = (\alpha(\boldsymbol{\Gamma_i}))_{i=1,\dots,k}$.

For the reader familiar with diverse groups [CS02], the generic framework is very similar to a diverse group, but with more structure: a vectorial space instead of a simple group. If we suppose θ is the identity function, $(\mathcal{X}^*, \mathcal{X}, \mathscr{L}, \mathbb{Z}_p)$ is a diverse group.

Example 1 (SPHF for DDH). Let us illustrate this framework for the DDH language: let g, h be two generators of a cyclic group $\mathbb{G}$ of prime order p, let $\mathcal{X} = \mathbb{G}^2$ and $\mathscr{L} = \{(g^r, h^r) \in \mathcal{X} \mid r \in \mathbb{Z}_p\}$. We set $\hat{\mathcal{X}} = \mathcal{X}$, $\hat{\mathscr{L}} = \mathscr{L}$ and θ the identify function so that $C = \hat{\boldsymbol{C}} = (u, v)$. $\hat{\mathscr{L}}$ is generated by the row vector $\boldsymbol{\Gamma_1} = (g, h)$. The witness for $C = (g^r, h^r)$ is $\lambda_1 = r$.

The hashing key $\mathsf{hk} = \alpha \stackrel{\$}{\leftarrow} \hat{\mathcal{X}}^*$ can be seen as a column vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2) \in \mathbb{Z}_p^2$ and

$$\begin{aligned}
\mathsf{hp} &= \gamma_1 = \alpha(\boldsymbol{\Gamma_1}) = \boldsymbol{\Gamma_1} \odot \boldsymbol{\alpha} = g^{\alpha_1} \cdot h^{\alpha_2} \\
H &= \alpha(\hat{\boldsymbol{C}}) = \hat{\boldsymbol{C}} \odot \boldsymbol{\alpha} = u^{\alpha_1} \cdot v^{\alpha_2} \\
&= r \odot \gamma_1 = \gamma_1^r.
\end{aligned}$$

This is exactly the original SPHF of Cramer and Shoup for the DDH language in [CS02].

Warm up: Conjunction of Languages. As a warm up, let us first construct the conjunction $\mathscr{L} = \mathscr{L}_1 \times \mathscr{L}_2$ of two languages $\mathscr{L}_1 \subset \mathcal{X}_1$ and $\mathscr{L}_2 \subset \mathcal{X}_2$ supported by the generic framework, in a more algebraic way than the one in [ACP09]. We can just set:

$$\begin{aligned} \hat{\mathcal{X}} &= \hat{\mathcal{X}}_1 \times \hat{\mathcal{X}}_2 & n &= n_1 + n_2 \\ \hat{\mathscr{L}} &= \hat{\mathscr{L}}_1 \times \hat{\mathscr{L}}_2 & k &= k_1 + k_2 \\ \theta((C_1, C_2)) = \hat{\boldsymbol{C}} &= (\theta_1(C_1), \theta_2(C_2)) & (\boldsymbol{\Gamma_i})_{i=1,\dots,k} &= ((\boldsymbol{\Gamma_i^{(1)}}, \boldsymbol{0})_{i=1,\dots,k_1}, (\boldsymbol{0}, \boldsymbol{\Gamma_i^{(2)}})_{i=1,\dots,k_2}) \end{aligned}$$

This is what is implicitly done in all conjunctions of SPHFs in [BBC+13], for example.

Example 2 (SPHF for Conjunction of DDH). Let g_1, h_1, g_2, h_2 be four generators of a cyclic group $\mathbb{G}$ of prime order p. Let $\mathcal{X}_1 = \mathcal{X}_2 = \mathbb{G}^2$ and $\mathscr{L}_i = \{(g_i^{r_i}, h_i^{r_i}) \in \mathcal{X}_i \mid r_i \in \mathbb{Z}_p\}$ for $i = 1, 2$. We set $\hat{\mathcal{X}}_i = \mathcal{X}_i$, $\hat{\mathscr{L}}_i = \mathscr{L}_i$ and θ_i the identify function so that $C_i = \hat{\boldsymbol{C}}_i = (u_i, v_i)$, for $i = 1, 2$. $\hat{\mathscr{L}}_i$ is generated by the row vector $\boldsymbol{\Gamma_1^{(i)}} = (g_i, h_i)$. The witness for $C_i = (g_i^{r_i}, h_i^{r_i})$ is $\lambda_1^{(i)} = r_i$.

Then, the SPHF for the conjunction of $\mathscr{L}_1$ and $\mathscr{L}_2$ is defined by:

$$\begin{aligned} \hat{\mathcal{X}} &= \hat{\mathcal{X}}_1 \times \hat{\mathcal{X}}_2 = \mathbb{G}^4 \\ \hat{\mathscr{L}} &= \hat{\mathscr{L}}_1 \times \hat{\mathscr{L}}_2 = \{(g_1^{r_1}, h_1^{r_1}, g_2^{r_2}, h_2^{r_2}) \mid r_1, r_2 \in \mathbb{Z}_p\} \\ \boldsymbol{\Gamma_1} &= (g_1, h_1, 1, 1) \in \mathbb{G}^4 \\ \boldsymbol{\Gamma_2} &= (1, 1, g_2, h_2) \in \mathbb{G}^4 \\ \theta(C) = \hat{\boldsymbol{C}} &= (\hat{\boldsymbol{C}}_1, \hat{\boldsymbol{C}}_2) = (u_1, v_1, u_2, v_2) \in \mathbb{G}^4 \qquad \text{for } C = (C_1, C_2) = ((u_1, v_1), (u_2, v_2)) \end{aligned}$$

The hashing key $\mathsf{hk} = \alpha \xleftarrow{\$} \hat{\mathcal{X}}^*$ can be seen as a column vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in \mathbb{Z}_p^4$ and

$$\begin{aligned} \mathsf{hp} &= (\gamma_1, \gamma_2) = (\boldsymbol{\Gamma_1} \odot \boldsymbol{\alpha}, \boldsymbol{\Gamma_2} \odot \boldsymbol{\alpha}) = (g_1^{\alpha_1} \cdot h_1^{\alpha_2}, g_2^{\alpha_3} \cdot h_2^{\alpha_4}) \\ H &= \alpha(\hat{\boldsymbol{C}}) = \hat{\boldsymbol{C}} \odot \boldsymbol{\alpha} = u_1^{\alpha_1} \cdot v_1^{\alpha_2} \cdot u_2^{\alpha_3} \cdot v_2^{\alpha_4} \\ &= r_1 \odot \gamma_1 \oplus r_2 \odot \gamma_2 = \gamma_1^{r_1} \cdot \gamma_2^{r_2}. \end{aligned}$$

Disjunction of Languages. We first remark we cannot naively extend the previous construction by choosing $\hat{\mathcal{X}} = \hat{\mathcal{X}}_1 \times \hat{\mathcal{X}}_2$ and $\hat{\mathscr{L}} = (\hat{\mathscr{L}}_1 \times \hat{\mathcal{X}}_2) \cup (\hat{\mathcal{X}}_1 \times \hat{\mathscr{L}}_2)$, because, in this case $\hat{\mathscr{L}}$ is not a subspace, and the subspace generated by $\hat{\mathscr{L}}$ is $\hat{\mathcal{X}}_1 \times \hat{\mathcal{X}}_2$. That is why we need to use tensor products of vector spaces instead of direct product of vector spaces. Concretely, we set

$$\begin{aligned} \hat{\mathcal{X}} &= \hat{\mathcal{X}}_1 \otimes \hat{\mathcal{X}}_2 & n &= n_1 \times n_2 \\ \hat{\mathscr{L}} &= \langle (\hat{\mathscr{L}}_1 \otimes \hat{\mathcal{X}}_2) \cup (\hat{\mathcal{X}}_1 \otimes \hat{\mathscr{L}}_2) \rangle & k &= k_1 \times n_2 + n_1 \times k_2 \\ \theta(C) = \hat{\boldsymbol{C}} &= \hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2 \end{aligned}$$

where $\langle V \rangle$ is the vector space generated by V. The vectors $\boldsymbol{\Gamma_i}$ are described in detail in the core of the paper. This construction works since, if $\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2 \in \hat{\mathscr{L}}$ then, thanks to properties of the tensor product, $\hat{\boldsymbol{C}}_1 \in \hat{\mathscr{L}}_1$ or $\hat{\boldsymbol{C}}_2 \in \hat{\mathscr{L}}_2$.

It is important to remark that computing a tensor product implies computing a multiplication. So if $\hat{\boldsymbol{C}}_1$ in $\hat{\mathcal{X}}_1$ and $\hat{\boldsymbol{C}}_2$ in $\hat{\mathcal{X}}_2$ are over some cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$, we need a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ to actually be able to compute $\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2$. More generally, doing the disjunction of K languages over cyclic groups requires a K-way multilinear map. This can be seen in the following example and we formally deal with this technicality in the core of the paper.

Example 3 (SPHF for Disjunction of DDH). Let us use the same notation as in Example 2, except that this time $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ is an asymmetric bilinear group (e is a bilinear map: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$), g_1, h_1 are generator of $\mathbb{G}_1$, g_2, h_2 are generators of $\mathbb{G}_2$, and $\mathcal{X}_i = \hat{\mathcal{X}}_i = \mathbb{G}_i^2$ (instead of $\mathbb{G}^2$) for $i = 1, 2$.

The disjunction of $\mathscr{L}_1$ and $\mathscr{L}_2$ is defined by

$$\begin{aligned}
\hat{\mathcal{X}} &= \hat{\mathcal{X}}_1 \otimes \hat{\mathcal{X}}_2 = \mathbb{G}_T^4 \\
k &= 4 \\
\hat{\mathscr{L}} &= \langle (\hat{\mathscr{L}}_1 \otimes \hat{\mathcal{X}}_2) \cup (\hat{\mathcal{X}}_1 \otimes \hat{\mathscr{L}}_2) \rangle \\
\boldsymbol{\Gamma_1} &= (g_1, h_1) \otimes (1 \in \mathbb{Z}_p, 0 \in \mathbb{Z}_p) = (g_1^1, g_1^0, h_1^1, h_1^0) = (g_1, 1, h_1, 1) \in \mathbb{G}_1^4 \\
\boldsymbol{\Gamma_2} &= (g_1, h_1) \otimes (0 \in \mathbb{Z}_p, 1 \in \mathbb{Z}_p) = (g_1^0, g_1^1, h_1^0, h_1^1) = (1, g_1, 1, h_1) \in \mathbb{G}_1^4 \\
\boldsymbol{\Gamma_3} &= (1 \in \mathbb{Z}_p, 0 \in \mathbb{Z}_p) \otimes (g_2, h_2) = (g_2^1, h_2^1, g_2^0, h_2^0) = (g_2, h_2, 1, 1) \in \mathbb{G}_2^4 \\
\boldsymbol{\Gamma_4} &= (0 \in \mathbb{Z}_p, 1 \in \mathbb{Z}_p) \otimes (g_2, h_2) = (g_2^0, h_2^0, g_2^1, h_2^1) = (1, 1, g_2, h_2) \in \mathbb{G}_2^4 \\
\theta(C) &= \hat{\boldsymbol{C}} = \hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2 = (u_1 \odot u_2, u_1 \odot v_2, v_1 \odot u_2, v_1 \odot v_2) \\
&= (e(u_1, u_2), e(u_1, v_2), e(v_1, u_2), e(v_1, v_2)) \in \mathbb{G}_T^4,
\end{aligned}$$

for $C = (C_1, C_2) = ((u_1, v_1), (u_2, v_2))$. The generating family of $\hat{\mathscr{L}}$ we used here is $(\boldsymbol{\Gamma_1}, \boldsymbol{\Gamma_2}, \boldsymbol{\Gamma_3}, \boldsymbol{\Gamma_4})$. As seen after, if we know the witness r_1 for C_1, we can use $\boldsymbol{\Gamma_1}$ and $\boldsymbol{\Gamma_2}$ to compute the hash value of $C = (C_1, C_2)$, while if we know the witness r_2 for C_2, we can use $\boldsymbol{\Gamma_3}$ and $\boldsymbol{\Gamma_4}$ to compute the hash value of C. Obviously this generating family is not free, since $\hat{\mathscr{L}}$ has dimension 3 and this family has cardinal 4.

The hashing key $\mathsf{hk} = \alpha \xleftarrow{\$} \hat{\mathcal{X}}^*$ can be seen as a column vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in \mathbb{Z}_p^4$ and

$$\begin{aligned}
\mathsf{hp} &= (\gamma_1, \gamma_2, \gamma_3, \gamma_4) = (g_1^{\alpha_1} \cdot h_1^{\alpha_3}, g_1^{\alpha_2} \cdot h_1^{\alpha_4}, g_2^{\alpha_1} \cdot h_2^{\alpha_2}, g_2^{\alpha_3} \cdot h_2^{\alpha_4}) \in \mathbb{G}_1^2 \times \mathbb{G}_2^2 \\
H &= \alpha(\hat{\boldsymbol{C}}) = \hat{\boldsymbol{C}} \odot \boldsymbol{\alpha} = e(u_1, u_2)^{\alpha_1} \cdot e(u_1, v_2)^{\alpha_2} \cdot e(v_1, u_2)^{\alpha_3} \cdot e(v_1, v_2)^{\alpha_4} \\
&= \begin{cases} r_1 \odot u_2 \odot \gamma_1 \oplus r_1 \odot v_2 \odot \gamma_2 = e(\gamma_1, u_2)^{r_1} \cdot e(\gamma_2, v_2)^{r_1} & \text{if } u_1 = g_1^{r_1} \text{ and } v_1 = h_1^{r_1} \\ r_2 \odot u_1 \odot \gamma_3 \oplus r_2 \odot v_1 \odot \gamma_4 = e(u_1, \gamma_3)^{r_2} \cdot e(v_1, \gamma_4)^{r_2} & \text{if } u_2 = g_2^{r_2} \text{ and } v_2 = h_2^{r_2}. \end{cases}
\end{aligned}$$

The last equalities, which show the way the projection hashing works, explain the choice of the generating family $(\boldsymbol{\Gamma_i})_i$.

2.2 Main Application: One-Time Simulation-Sound NIZK Arguments

The language of the NIZK is $\mathscr{L}_1$, while $\mathscr{L}_2$ is a hard subset membership language used to build the NIZK. For the sake of simplicity, we suppose that $\mathscr{L}_2 = \hat{\mathscr{L}}_2$, $\mathcal{X}_2 = \hat{\mathcal{X}}_2$, and θ_2 is the identity function. We will consider the SPHF of the disjunction of $\mathscr{L}_1$ and $\mathscr{L}_2$, so we need to suppose that it is possible to do it. For this high level overview, let us just suppose that $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ is a bilinear group and that $\mathscr{L}_1$ is defined over $\mathbb{G}_1$, $\mathscr{L}_2$ over $\mathbb{G}_2$. If DDH holds in $\mathbb{G}_2$, $\mathscr{L}_2$ can just be the DDH language in $\mathbb{G}_2$ recalled in Example 1.

The common reference string is a projection key hp for the disjunction of $\mathscr{L}_1$ and $\mathscr{L}_2$, while the trapdoor (to simulate proofs) is the hashing key. Essentially, a proof for a statement C_1 is just the hash values of $(C_1, \boldsymbol{e_{2,i_2}})$ where $(\boldsymbol{e_{2,i_2}})_{i_2}$ are the scalar vectors of the canonical base of $\hat{\mathcal{X}}_2$.

The basic idea is that a valid proof for a word $C_1 \in \mathscr{L}_1$ enables to compute the hash value H' of (C_1, C_2) for any word $C_2 \in \hat{\mathcal{X}}_2$, by linearly combining elements of the proof, since any word C_2 can be written as a linear combination of $(\boldsymbol{e_{2,i_2}})_{i_2}$. Therefore to check a proof, we just need to check that the hash value H' for the words $(C_1, \boldsymbol{\Gamma_j^{(2)}})$, for $j = 1, \ldots, k_2$, computed as above, are equal to the hash value H of the same words computed using hp and the witness for $\boldsymbol{\Gamma_j^{(2)}} \in \mathscr{L}_2$. Such a witness is trivial and can be efficiently computed by anyone, since a witness for a word C_2 consists in the coefficients of a linear combination of $(\boldsymbol{\Gamma_i^{(2)}})_{i=1,\ldots,k_2}$ equal to C_2. More precisely, it is a vector of the form $(0, \ldots, 0, 1, 0, \ldots, 0) \in \mathbb{Z}_p^{k_2}$.

The trapdoor, i.e., the hashing key, clearly enables to simulate any proof, and the resulting proofs are perfectly indistinguishable from normal ones, hence the perfect zero-knowledge property. Moreover, the soundness comes from the fact that a proof for a word $C_1 \notin \mathscr{L}_1$ can be used to break the hard subset membership in $\mathbb{G}_2$.

More precisely, on the one hand, a valid proof for any word $C_1 \in \mathcal{X}_1$ enables to compute the hash value H' of (C_1, C_2) for any word $C_2 \in \mathscr{L}_2$, by linearly combining elements of the proofs, and the validity of the proof ensures H' is correct if $C_2 \in \mathscr{L}_2$. On the other hand, we can also compute a hash value H of (C_1, C_2) for any $C_2 \in \mathcal{X}_2$ using the hashing key hk. Then, if $C_2 \in \mathscr{L}_2$, necessarily $H = H'$, while if $C_2 \notin \mathscr{L}_2$, H looks completely random when given only hp. Since H' does not depend on hk but only on hp, it is different from H with overwhelming probability. Hence the soundness.

Example 4 (NIZK for DDH in $\mathbb{G}_1$). Using the SPHF in Example 3, the proof for a word $C_1 = (u_1 = g_1^r, v_1 = h_1^r) \in \mathbb{G}_1^2$ is the vector $\boldsymbol{\pi} = (\pi_1, \pi_2) \in \mathbb{G}_1^2$ where: π_1 is the hash value of $(C_1, (1, 0)) \in \mathbb{G}_1^2 \times \mathbb{Z}_p^2$ and π_2 is the hash value of $(C_1, (0, 1)) \in \mathbb{G}_1^2 \times \mathbb{Z}_p^2$. Concretely we have:

$$\pi_1 = r \odot \gamma_1 = \gamma_1^r \in \mathbb{G}_1 \qquad\qquad \pi_2 = r \odot \gamma_2 = \gamma_2^r \in \mathbb{G}_1.$$

This proof is valid if and only if:

$$e(\pi_1, g_2) \cdot e(\pi_2, h_2) = \pi_1 \odot g_2 \oplus \pi_2 \odot h_2 \stackrel{?}{=} u_1 \odot \gamma_3 \oplus v_1 \odot \gamma_4 = e(u_1, \gamma_3) \cdot e(v_1, \gamma_4).$$

This check can be done using the common reference string $\mathsf{hp} = (\gamma_1, \gamma_2, \gamma_3, \gamma_4)$.

Finally, to simulate a proof for $C_1 = (u_1, v_1)$ without knowing any witness for C_1 but knowing the trapdoor $\mathsf{hk} = \boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in \mathbb{Z}_p^4$, we compute π_1 and π_2 as follows:

$$\pi_1 \coloneqq \alpha_1 \odot u_1 \oplus \alpha_3 \odot v_1 = u_1^{\alpha_1} \cdot v_1^{\alpha_3} \qquad\qquad \pi_2 \coloneqq \alpha_2 \odot u_1 \oplus \alpha_4 \odot v_1 = u_1^{\alpha_2} \cdot v_1^{\alpha_4}.$$

To get a one-time simulation-sound NIZK, we replace the SPHF over $\mathscr{L}_1$ by a stronger kind of SPHF for which, roughly speaking, the hash value of a word $C \notin \mathscr{L}_1$ appears random even if we are given the projection key hp and the hash value of another word $C \in \mathcal{X}_1$ of our choice. We show that it is always possible to transform a normal SPHF into this stronger variant, assuming the existence of collision-resistant hash functions[2].

2.3 Other Applications

TSPHF. Trapdoor SPHF is a direct application of disjunctions of SPHFs: as for NIZK, the language we are interested in is $\mathscr{L}_1$, while $\mathscr{L}_2$ is a hard subset membership language. The common reference string contains a word $C_2 \in \mathscr{L}_2$, and the trapdoor is just a witness w_2 for this word. The hash value of some $C_1 \in \mathcal{X}_1$, is the hash value of (C_1, C_2) for the disjunction of $\mathscr{L}_1$ and $\mathscr{L}_2$, which can be computed in two or three ways: using hk, or using hp and w_1 (classical projection hashing — possible only when $C_1 \in \mathscr{L}_1$ and w_1 is a witness for it), or using hp and w_2 (trapdoor). The smoothness comes from the hard subset membership property of $\mathscr{L}_2$ (which says that this common reference string is indistinguishable from a word $C_2 \notin \mathscr{L}_2$) and the fact that when $C_2 \notin \mathscr{L}_2$, the hash value of (C_1, C_2) appears random by smoothness when $C_1 \notin \mathscr{L}_1$, given only hp.

One-Round GPAKE. The construction extends the one-round PAKE of Benhamouda et al. in [BBC+13], which is an efficient instantiation of the Katz-Vaikuntanathan framework [KV11]. Basically, a user U_i sends an extractable commitment C_i (i.e., an encryption for some public key ek in the common reference string) of his password pw together with a projection key hp_i for the

[2] Actually, the use of collision-resistant hash functions could be avoided, but that would make the construction much less efficient.

disjunction of $n-1$ languages of valid commitments of pw (words in this disjunction are tuple $\boldsymbol{C}_i = (C_j)_{j \neq i}$ of $n-1$ commitments where at least one of them is a valid commitment of pw). Each partner U_j of this user U_i can compute the hash value H_i of the tuple $\boldsymbol{C}_i$, with hp_i, just by additionally knowing the witness (the random coins) of his commitment C_j onto pw, while U_i uses hk_i. The resulting secret key K is just the XOR of all these hash values (one per hashing key, i.e., one per user): $K = \bigoplus_i H_i$.

At a first glance, one may wonder why our construction relies on a disjunction and not on a conjunction: intuitively, as a user, we would like that every other users committed to the same password as ours. Unfortunately, in this case, nobody would be able to compute the hash value of the expected conjunction, except the user who generated the hashing key. This is because this computation would require knowing all the witnesses and there is no way for a user to know the witness for a commitment of another user. However, by relying on a disjunction, each user is only required to know the witness for his own commitment.

To understand why this is a secure solution, please note that the challenger (in the security game) can make dummy commitments for the honest players he controls. Then, if no corrupted user (controlled by the adversary) commits to a correct password, the tuple of the $n-1$ other commitments would not be a valid word in the disjunction language (no commitment would be valid) for any of the honest users. Hence, the hash value would appear random to the adversary.

2.4 Pseudo-Random Projective Hash Functions and More Efficient Applications

Pseudo-Random Projective Hash Functions. As already explained in Section 1.1, for our (one-time simulation-sound) NIZK and our TSPHF, the second language $\mathscr{L}_2$ is used to provide extra features. Security properties come from its hard subset membership property.

However, hard subset membership comes at a cost: the dimension k_2 of $\hat{\mathscr{L}}_2$ has to be at least 1 to be non-trivial, and so the dimension n_2 of $\hat{\mathcal{X}}_2$ is at least 2, otherwise $\hat{\mathscr{L}}_2 = \hat{\mathcal{X}}_2$. This makes the projection key of the disjunction of $\mathscr{L}_1$ and $\mathscr{L}_2$ of size $k_1 n_2 + n_1 k_2 \geq 2k_1 + n_1$.

Intuitively, what we would like is to be able to have a language $\mathscr{L}_2$ where $n_2 = k_2 = 1$. Such a language would clearly not be hard subset membership, and the smoothness property of SPHF would be completely trivial, since $\hat{\mathcal{X}}_2 \setminus \hat{\mathscr{L}}_2$ would be empty. That is why we introduce the notion of pseudo-randomness which says that the hash value of a word C_2 chosen at random in $\mathcal{X}_2$ (and for implicit languages parameters chosen at random), the hash value of C_2 looks random, given only the projection key.

Under DDH in $\mathbb{G}_2$, we can simply choose $\mathsf{crs}_2 = g_2$ a random generator in $\mathbb{G}_2$, $\mathcal{X}_2 = \hat{\mathcal{X}}_2 = \mathscr{L}_2 = \hat{\mathscr{L}}_2 = \mathbb{G}_2$, and θ_2 the identity function. The witness for a word $C_2 \in \mathbb{G}_2$ is just its discrete logarithm in base g_2, and so $\hat{\mathscr{L}}_2$ is seen as generated by the vector $\boldsymbol{\Gamma}^{\mathbf{(2)}}_{\mathbf{1}} = (g_2)$. An hashing key hk is just a random scalar $\alpha \in \mathbb{Z}_p$, the associated projection key is $\mathsf{hp} = g_2^\alpha$. Finally the hash value is $H = C_2^\alpha$. It can also be computed using hp if we know the discrete logarithm of C_2. The DDH assumption says that if $g_2, \mathsf{hp} = g_2^\alpha, C_2$ are chosen uniformly at random in $\mathbb{G}_2$, it is hard to distinguish $H = C_2^\alpha$ from a random group element $H \in \mathbb{G}_2$; hence the pseudo-randomness.

Mixed Pseudo-Randomness. In all our applications, we are not really interested in the SPHF on $\mathscr{L}_2$ but in the SPHF on the disjunction $\mathscr{L}$ of $\mathscr{L}_1$ and $\mathscr{L}_2$. Of course this SPHF would be smooth, but that property is again trivial, since all words (C_1, C_2) are in $\mathscr{L}$. We therefore again need a stronger property called *mixed pseudo-randomness* which roughly says that if hk is a random hashing key, if $C_1 \notin \mathscr{L}_1$ and if C_2 is chosen at random, the hash value of $(C_1, C_2) \in \mathscr{L}$ appears random to any polynomial-time adversary, even given access to the projection key hp.

The proof of this property is very technical and requires that it is possible to generate parameters of $\mathscr{L}_1$ so that we know the discrete logarithm of the generators $(\boldsymbol{\Gamma}^{\mathbf{(1)}}_{\boldsymbol{i_1}})_{i_1}$ of $\hat{\mathscr{L}}_1$. This last property is verified by most, if not all, languages we are interested in.

Applications. Using the mixed pseudo-randomness property, we easily get more efficient NIZK and TSPHF, just by replacing $\mathscr{L}_2$ by a language $\mathscr{L}_2$ with a pseudo-random Projective Hash Function. Getting a more efficient one-time simulation-sound NIZK is slightly more complex and is only detailed in the core of the paper.

3 Preliminaries

3.1 Notations

As usual, all the players and the algorithms will be possibly probabilistic and stateful. Namely, adversaries can keep a state st during the different phases, and we denote $\stackrel{\$}{\leftarrow}$ the outcome of a probabilistic algorithm or the sampling from a uniform distribution. For example, $\mathcal{A}(x;r)$ will denote the execution of $\mathcal{A}$ with input x and random tape r. For the sake of clarity, sometimes, the latter random tape will be dropped, with the notation $\mathcal{A}(x)$.

The qualities of adversaries will be measured by their successes and advantages in certain experiments $\mathsf{Exp}^{\mathsf{sec}}$ or $\mathsf{Exp}^{\mathsf{sec}-b}$ (between the cases $b=0$ and $b=1$), denoted $\mathsf{Succ}^{\mathsf{sec}}(\mathcal{A},\mathfrak{K})$ and $\mathsf{Adv}^{\mathsf{sec}}(\mathcal{A},\mathfrak{K})$ respectively, where $\mathfrak{K}$ is the security parameter. Formal definition of all of this and of statistical distance can be found in Appendix A.1.

3.2 Definition of SPHF

Let $(\mathscr{L}_{\mathsf{crs},\mathsf{par}})_{\mathsf{crs},\mathsf{par}}$ be a family of NP languages indexed by $(\mathsf{crs},\mathsf{par})$ with witness relation $\mathcal{R}_{\mathsf{crs},\mathsf{par}}$, namely $\mathscr{L}_{\mathsf{crs},\mathsf{par}} = \{x \in \mathcal{X}_{\mathsf{crs}} \mid \exists w,\ \mathcal{R}_{\mathsf{crs},\mathsf{par}}(x,w) = 1\}$, where $(\mathcal{X}_{\mathsf{crs}})_{\mathsf{crs}}$ is a family set. crs is generated by some polynomial-time algorithm $\mathsf{Setup}_{\mathsf{crs}}$ taking as input the unary representation of the security parameter λ, while par is an arbitrary parameter in $\mathsf{Params}_{\mathsf{crs}}$. We suppose that membership to $\mathcal{X}_{\mathsf{crs}}$ and $\mathcal{R}_{\mathsf{crs},\mathsf{par}}$ can be check in polynomial time (in $\mathfrak{K}$).

Finally, we suppose that $\mathsf{Setup}_{\mathsf{crs}}$ also outputs a trapdoor $\mathcal{T}_{\mathsf{crs}}$ associated to crs. This trapdoor is empty $\perp$ in most cases, but for some applications, we require that $\mathcal{T}_{\mathsf{crs}}$ contains enough information to decide whether a word $C \in \mathcal{X}$ is in $\mathscr{L}$ or not (or slightly more information). We notice that for most, if not all, languages (we are interested in), it is easy to make $\mathsf{Setup}_{\mathsf{crs}}$ output such a trapdoor, without changing the distribution of crs.

An SPHF over $(\mathscr{L}_{\mathsf{crs},\mathsf{par}})$ is defined by four polynomial-time algorithms:

- $\mathsf{HashKG}(\mathsf{crs})$ generates a hashing key hk;
- $\mathsf{ProjKG}(\mathsf{hk},\mathsf{crs})$ derives a projection key hp from hk;
- $\mathsf{Hash}(\mathsf{hk},(\mathsf{crs},\mathsf{par}),C)$ outputs the hash value from the hashing key, for any $(\mathsf{crs},\mathsf{par})$ and for any word $C \in \mathcal{X}$;
- $\mathsf{ProjHash}(\mathsf{hp},(\mathsf{crs},\mathsf{par}),C,w)$ outputs the hash value from the projection key hp, and the witness w, for a word $C \in \mathscr{L}_{\mathsf{crs},\mathsf{par}}$ ($\mathcal{R}_{\mathsf{crs},\mathsf{par}}(C,w)=1$).

The set of hash values is called the *range* of the SPHF and is denoted Π. It is often a cyclic group. We always suppose that its size is superpolynomial in the security parameter $\mathfrak{K}$ so that the probability to guess correctly a uniform hash value is negligible.

A SPHF has to verify two properties:

- *Correctness.* For any crs, any par and any word $C \in \mathscr{L}_{\mathsf{crs},\mathsf{par}}$ with witness w (i.e., such that $\mathcal{R}_{(\mathsf{crs},\mathsf{par})}(C,w)=1$), for any $\mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathsf{crs})$ and for $\mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk},\mathsf{crs})$,
$$\mathsf{Hash}(\mathsf{hk},(\mathsf{crs},\mathsf{par}),C) = \mathsf{ProjHash}(\mathsf{hp},(\mathsf{crs},\mathsf{par}),C,w);$$
- *Smoothness.* The hash value of a word outside the language looks completely random. More precisely, an SPHF is ε-smooth if for any crs, and any function (not necessarily computable

in polynomial time) f from the set of projection keys to $\mathsf{Params}_{\mathsf{crs}} \times \mathcal{X}_{\mathsf{crs}}$, so that $(\mathsf{par}, C) = f(\mathsf{hp})$ is such that $C \notin \mathscr{L}_{(\mathsf{crs},\mathsf{par})}$, the two following distributions are ε-statistically close:

$$\left\{(\mathsf{hp}, H) \middle| \begin{array}{ll} \mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathsf{crs}); & \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{crs}); \\ (\mathsf{par}, C) \leftarrow f(\mathsf{hp}); & H \leftarrow \mathsf{Hash}(\mathsf{hk}, (\mathsf{crs}, \mathsf{par}), C) \end{array}\right\}$$

$$\left\{(\mathsf{hp}, H) \middle| \begin{array}{ll} \mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathsf{crs}); & \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{crs}); \\ (\mathsf{par}, C) \leftarrow f(\mathsf{hp}); & H \stackrel{\$}{\leftarrow} \Pi \end{array}\right\}.$$

An SPHF is smooth if it is ε-smooth with ε negligible (in $\mathfrak{K}$). Actually, in this article, all SPHFs are even 0-smooth. An equivalent and simpler way to define 0-smoothness is the following: for any crs, any par and any $C \notin \mathscr{L}_{(\mathsf{crs},\mathsf{par})}$, the following two distributions are identical:

$$\left\{(\mathsf{hp}, H) \mid \mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathsf{crs}); \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{crs}); H \leftarrow \mathsf{Hash}(\mathsf{hk}, (\mathsf{crs}, \mathsf{par}), C)\right\}$$

$$\left\{(\mathsf{hp}, H) \mid \mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathsf{crs}); \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{crs}); H \stackrel{\$}{\leftarrow} \Pi\right\}.$$

Indeed, since the two distributions are identical, the fact C and par can depend on hp gives no advantage.

This definition of SPHF actually corresponds to a strong version of $\mathsf{KV\text{-}SPHF}$ [BBC+13], where hp does not depend on par nor on C, and the smoothness holds even if C is chosen depending on hp (via this function f). In particular, it is stronger than the definition of SPHF given in [CS02], where the smoothness is actually defined only for random elements $C \in \mathcal{X} \setminus \mathscr{L}_{\mathsf{crs},\mathsf{par}}$. This is also slightly stronger than the 1-universal hash proof systems also defined in [CS02], since the hash value is supposed to look completely random and not just having some minimal entropy.

In the sequel, crs is often dropped to simplify notations.

3.3 Hard Subset Membership Languages

A family of languages $(\mathscr{L}_{\mathsf{crs},\mathsf{par}} \subseteq \mathcal{X}_{\mathsf{crs}})_{\mathsf{crs},\mathsf{par}}$ is said to be a hard subset membership family of languages, if is hard to distinguish between a word randomly drawn from inside $\mathscr{L}_{\mathsf{crs},\mathsf{par}}$ from a word randomly drawn from outside $\mathscr{L}_{\mathsf{crs},\mathsf{par}}$ (i.e., from $\mathcal{X}_{\mathsf{crs}} \setminus \mathscr{L}_{\mathsf{crs},\mathsf{par}}$). This definition implicitly assumes the existence of a distribution over $\mathcal{X}_{\mathsf{crs}}$ and a way to sample efficiently words from $\mathscr{L}_{\mathsf{crs},\mathsf{par}}$ and from $\mathcal{X}_{\mathsf{crs}} \setminus \mathscr{L}_{\mathsf{crs},\mathsf{par}}$. More precisely, this property is defined by the experiments $\mathsf{Exp}^{\texttt{subset-memb-}b}$ depicted in Fig. 1.

$\mathsf{Exp}^{\texttt{subset-memb-}b}(\mathcal{A}, \mathfrak{K})$
  $(\mathsf{crs}, \mathcal{T}_{\mathsf{crs}}) \stackrel{\$}{\leftarrow} \mathsf{Setup}_{\mathsf{crs}}(1^{\mathfrak{K}})$
  $(\mathsf{par}, \mathsf{st}) \stackrel{\$}{\leftarrow} \mathcal{A}(\mathsf{crs})$
  if $b = 0$ **then**
    $C \stackrel{\$}{\leftarrow} \mathscr{L}_{\mathsf{crs},\mathsf{par}}$
  else
    $C \stackrel{\$}{\leftarrow} \mathcal{X}_{\mathsf{crs}} \setminus \mathscr{L}_{\mathsf{crs},\mathsf{par}}$
  return $\mathcal{A}(\mathsf{st}, C)$

Fig. 1. Experiments $\mathsf{Exp}^{\texttt{subset-memb-}b}$ for hard subset membership

3.4 Bilinear Groups

All our concrete constructions are based on bilinear groups, which are extensions of cyclic groups.

Cyclic Groups. $(p, \mathbb{G}, g)$ denotes a cyclic group $\mathbb{G}$ of order p and of generator g.

Bilinear Groups. 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 group* if $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ 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 group 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 group.

Assumptions. The assumption we use the most is the SXDH assumption The SXDH assumption over a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ says the DDH assumption holds both in $(p, \mathbb{G}_1, g_1)$ and $(p, \mathbb{G}_2, g_2)$, where the DDH assumption is defined as follows:

Definition 5 (Decisional Diffie-Hellman (DDH)). *The Decisional Diffie-Hellman assumption says that, in a cyclic 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 also propose constructions under weaker assumptions than SXDH or DDH, namely κ-Lin, defined as follows:

Definition 6 (κ-Lin). *The κ-Linear assumption says that, in a cyclic group $(p, \mathbb{G}, g)$, when we are given $(g^{a_1}, \dots, g^{a_\kappa}, g^{a_1 b_1}, \dots, g^{a_\kappa b_\kappa}, g^c)$ for unknown $a_1, \dots, a_\kappa, b_1, \dots, b_\kappa \xleftarrow{\$} \mathbb{Z}_p$, it is hard to decide whether $c = b_1 + \cdots + b_\kappa$ (a κ-Lin tuple) or $c \xleftarrow{\$} \mathbb{Z}_p$ (a random tuple).*

One advantage of 2-Lin in particular is that it can hold even on symmetric bilinear groups while DDH or SXDH do not. We often say κ-Lin for κ-Lin in $\mathbb{G}_1$ and in $\mathbb{G}_2$.

4 Smooth Projective Hash Functions for Disjunctions

4.1 Graded Rings

Let us first recall the notion of graded ring introduced in [BBC+13]. Graded rings are a generalization of bilinear groups and can be used as a practical abstraction of multi-linear maps coming from the framework of Garg, Gentry and Halevi [GGH13].

Although, in this article, we focus on bilinear groups, all we do can easily be extended to multilinear groups, based on multilinear maps of Coron et al. [CLT13]. In addition, graded rings help simplifying notations.

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

$$\sup(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{v}}') = (\max(\tilde{\boldsymbol{v}}_1, \tilde{\boldsymbol{v}}'_1), \dots, \max(\tilde{\boldsymbol{v}}_\tau, \tilde{\boldsymbol{v}}'_\tau)) \quad \inf(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{v}}') = (\min(\tilde{\boldsymbol{v}}_1, \tilde{\boldsymbol{v}}'_1), \dots, \min(\tilde{\boldsymbol{v}}_\tau, \tilde{\boldsymbol{v}}'_\tau)).$$

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

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

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

We remark that $\odot$ is only a partial binary operation and we use the following convention: $\perp \oplus u = u \oplus \perp = u \odot \perp = \perp \odot u = \perp$, for any $u \in \mathfrak{G} \cup \{\perp\}$. Let $\mathfrak{G}_{\tilde{\boldsymbol{v}}}$ be the additive group $\{u = [\tilde{\boldsymbol{v}}', x] \in \mathfrak{G} \mid \tilde{\boldsymbol{v}}' = \tilde{\boldsymbol{v}}\}$ of graded ring elements of index $\tilde{\boldsymbol{v}}$.

We will make natural use of vector and matrix operations over graded ring elements. In particular, we say that $\mathfrak{G}^n$ and $\mathfrak{G}^{1\times n}$ are *vector spaces* over the graded ring $\mathfrak{G}$. The canonical basis $(\boldsymbol{e_i})_{i=1,\ldots,n}$ of $\mathfrak{G}^{1\times n}$ is defined as usual, except the vectors of the canonical basis are of index $\bar{0}$ (i.e., can be considered as "scalars"). Finally, if F is a family of vectors, $\langle F \rangle$ denotes the vector space generated by F.

Sub-Graded Ring and Multiplicative Compatibility. A sub-graded ring of a graded ring $\mathfrak{G}$ is a subset $\mathfrak{G}_{\leq \tilde{\boldsymbol{v}}} = \{u = [\tilde{\boldsymbol{v}}', x] \in \mathfrak{G} \mid \tilde{\boldsymbol{v}}' \leq \tilde{\boldsymbol{v}}\}$ of $\mathfrak{G}$. A sub-graded ring is itself a graded ring (or more precisely, is isomorphic to a graded ring). Two sub-graded ring $\mathfrak{G}_1 = \mathfrak{G}_{\leq \tilde{\boldsymbol{v}}_1}$ and $\mathfrak{G}_2 = \mathfrak{G}_{\leq \tilde{\boldsymbol{v}}_2}$ are said *multiplicatively compatible* if $\tilde{\boldsymbol{v}}_1 + \tilde{\boldsymbol{v}}_2 \in \Lambda$, or in other words, if it is possible to multiply any element in $\mathfrak{G}_{\leq \tilde{\boldsymbol{v}}_1}$ by an element in $\mathfrak{G}_{\leq \tilde{\boldsymbol{v}}_2}$.

Cyclic Groups, Asymmetric Bilinear Groups, and Notations Let us now show that cyclic groups and bilinear groups of order p can be seen as graded rings over $R = \mathbb{Z}_p$:

Cyclic groups: $\kappa = \tau = 1$. More precisely, 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 $g^x \in \mathbb{G}$.

Asymmetric bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$: $\kappa = 1$ 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 $e(g_1, g_2)^x \in \mathbb{G}_T$.
The two non-trivial sub-graded rings of this bilinear group are $\mathfrak{G}_{\leq(1,0)}$ and $\mathfrak{G}_{\leq(0,1)}$. These two sub-graded rings are multiplicatively compatible, since $(1,0) + (0,1) = (1,1)$. By abuse of notation, we often call these sub-graded rings: $\mathbb{G}_1$ and $\mathbb{G}_2$.

Symmetric bilinear groups $(p, \mathbb{G}, \mathbb{G}_T, e, g)$: $\kappa = 2$ 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 $e(g, g)^x \in \mathbb{G}_T$.
The non-trivial sub-graded ring of this bilinear group is $\mathfrak{G}_{\leq 1}$. This sub-graded ring is multiplicatively compatible with itself, since $1 + 1 = 2 = \kappa$. By abuse of notation, we often call this sub-graded ring: $\mathbb{G}$.

We have chosen an additive notation for the group law in $\mathfrak{G}_{\tilde{\boldsymbol{v}}}$, due to the fact that it simplifies the description of our constructions in the generic framework. Unfortunately, this choice of notation also makes it is somewhat cumbersome when dealing with bilinear groups. Hence, 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. Therefore, 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:

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

4.2 Generic Framework and Diverse Vector Spaces

Let us now recall the generic framework for SPHFs. We have already seen the main ideas of this framework in Section 2.1. These ideas were stated in term of generic vector space. Even though using generic vector spaces facilitates the explanation of high level ideas, it is better to use explicit basis when it comes to details.

In addition, we now deal with families of languages $(\mathscr{L}_{\mathsf{par}})_{\mathsf{par}}$ instead of just a language. This means that we now have a family of functions $\theta_{\mathsf{par}} : \mathcal{X} \to \hat{\mathcal{X}}$ (so that $\theta_{\mathsf{par}}(C) \in \hat{\mathscr{L}}$ if and only if $C \in \mathscr{L}_{\mathsf{par}}$), instead of just a single function θ. It is important to notice that, since we do not want hp to depend on par. In addition, there are only one vector space $\hat{\mathcal{X}}$ and one subspace $\hat{\mathscr{L}}$ for all par.

Let $\mathfrak{G}$ be a graded ring. We now set $\hat{\mathcal{X}} = \mathfrak{G}^{1\times n}$, so that any vector $\hat{\boldsymbol{C}} \in \hat{\mathcal{X}}$ is a n-dimensional *row* vector. We denote by $(\boldsymbol{e_i})_{i=1,\dots,n}$ the canonical basis of $\hat{\mathcal{X}}$. The dual space of $\hat{\mathcal{X}}$ is isomorphic to $\mathbb{Z}_p^n$, and the hashing key $\alpha \in \hat{\mathcal{X}}$ corresponds to the *column* vector $\boldsymbol{\alpha} = (\alpha_i)_{i=1,\dots,n}$, with $\alpha_i = \alpha(\boldsymbol{e_i})$. Finally, we denote by Γ the matrix with rows $(\boldsymbol{\Gamma_i})_{i=1,\dots,k}$, where the family $(\boldsymbol{\Gamma_i})$ generate the subspace $\hat{\mathscr{L}}$ of $\hat{\mathcal{X}}$.

We suppose that, a word $C \in \mathcal{X}$ is in $\mathscr{L}_{\mathsf{par}}$ if and only if there exists $\boldsymbol{\lambda} \in \mathfrak{G}^{1\times k}$ such that:

$$\theta_{\mathsf{par}}(C) = \boldsymbol{\lambda} \odot \Gamma.$$

In addition, we suppose that $\boldsymbol{\lambda}$ can be computed easily from any witness w for C; and in the sequel we often simply consider that $w = \boldsymbol{\lambda}$.

By analogy with diverse groups [CS02], as explained in Section 2.1, we say that $\mathcal{V} = (\mathcal{X}, (\mathscr{L}_{\mathsf{par}}), \mathfrak{G}, n, k, \Gamma, (\theta_{\mathsf{par}}))$ verifying the above property is a *diverse vector space*.

To each diverse vector space $\mathcal{V}$, we can associate an SPHF as follows:

$$\mathsf{hk} := \boldsymbol{\alpha} \xleftarrow{\$} \mathbb{Z}_p^n \qquad \mathsf{hp} := \boldsymbol{\gamma} = \Gamma \odot \boldsymbol{\alpha} \qquad H := \hat{\boldsymbol{C}} \odot \boldsymbol{\alpha} = \boldsymbol{\gamma} \odot \boldsymbol{\lambda} =: H'$$

if $\hat{\boldsymbol{C}} = \theta_{\mathsf{par}}(C)$, and where the last equality holds when $C \in \mathscr{L}_{\mathsf{par}}$ and $\boldsymbol{\lambda}$ is a witness for that.

It is straighforward to see (and this is proven in [BBC+13]) that any SPHF defined by a discrete vector space $\mathcal{V}$ as above is correct and smooth.

4.3 Disjunctions of SPHFs

As explained in Section 2.1, an SPHF for the disjunction of two languages $\mathscr{L}_1$ and $\mathscr{L}_2$ roughly consists in a tensor products of their related vector spaces $\hat{\mathcal{X}}_1$ and $\hat{\mathcal{X}}_2$. However, our vector spaces are not classical vector spaces, since they are over graded rings. In particular, multiplication of scalars is not always possible, and so tensor product may not be always possible either. That is why, we first need to introduce the notion of tensor product of vector spaces over graded rings, before giving the detailed construction of disjunctions of SPHFs.

Tensor Product of Vector Spaces over Graded Rings. Let us very briefly recall notations for tensor products and adapt them to vector spaces over graded rings.

Let $\mathfrak{G}_1$ and $\mathfrak{G}_2$ be two multiplicatively compatible sub-graded ring of $\mathfrak{G}$. Let V_1 be a n_1-dimensional vector space over $\mathfrak{G}_1$ and V_2 be a n_2-dimensional vector space over $\mathfrak{G}_2$. Let $(\boldsymbol{e_{1,i}})_{i=1,\dots,n_1}$ and $(\boldsymbol{e_{2,i}})_{i=1,\dots,n_2}$ be basis of V_1 and V_2 respectively. Then the *tensor product* V of V_1 and V_2, denoted $V = V_1 \otimes V_2$ is the $n_1 \times n_2$-dimensional vector space over $\mathfrak{G}$ generated by the free family $(\boldsymbol{e_{1,i}} \otimes \boldsymbol{e_{2,j}})_{\substack{i=1,\dots,n_1 \\ j=1,\dots,n_2}}$. If $\boldsymbol{u} = \bigoplus_{i=1}^{n_1} u_i \odot \boldsymbol{e_{1,i}}$ and $\boldsymbol{v} = \bigoplus_{j=1}^{n_2} v_j \odot \boldsymbol{e_{2,j}}$, then:

$$\boldsymbol{u} \otimes \boldsymbol{v} = \bigoplus_{i=1}^{n_1} \bigoplus_{j=1}^{n_2} (u_i \odot v_j) \odot (e_{1,i} \otimes e_{2,j}).$$

In addition, $\otimes$ is a bilinear operator.

More generally, we can define the tensor product of two matrices $M \in \mathfrak{G}_1^{k\times m}$ and $M' \in \mathfrak{G}_2^{k'\times m'}$, $T = M \otimes M' \in \mathfrak{G}^{kk'\times mm'}$ by

$$T_{(i-1)k'+i',(j-1)m'+j'} = M_{i,j} \odot M'_{i',j'} \qquad \text{for } \begin{cases} i = 1,\dots,k,\ i' = 1,\dots,k', \\ j = 1,\dots,m,\ j' = 1,\dots,m'. \end{cases}$$

And if $M \in \mathfrak{G}_1^{k\times m}$, $M' \in \mathfrak{G}_2^{k'\times m'}$, $N \in \mathfrak{G}_1^{m\times n}$ and $N' \in \mathfrak{G}_2^{m'\times n'}$, and if $M \odot N$ and $M' \odot N'$ are well-defined (i.e., index of coefficients are "coherent"), then we have

$$(M \otimes M') \odot (N \otimes N') = (M \odot N) \otimes (M' \odot N').$$

Finally, this definition can be extended to more than 2 vector spaces.

Disjunctions of SPHFs. Let $\mathcal{V}_1 = (\mathcal{X}_1, (\mathscr{L}_{1,\mathsf{par}}), \mathfrak{G}_1, n_1, k_1, \Gamma^{(1)}, (\theta_{1,\mathsf{par}}))$ and $\mathcal{V}_2 = (\mathcal{X}_2, (\mathscr{L}_{2,\mathsf{par}}), \mathfrak{G}_2, n_2, k_2, \Gamma^{(2)}, (\theta_{2,\mathsf{par}}))$ be two diverse vector spaces over two multiplicatively sub-graded rings $\mathfrak{G}_1$ and $\mathfrak{G}_2$ of some graded ring $\mathfrak{G}$.

In applications, we will often have $\mathfrak{G}_1 = \mathbb{G}_1$ and $\mathfrak{G}_2 = \mathbb{G}_2$ where $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ is a bilinear group.

Let us set:

$$\begin{aligned} \mathcal{X} &= \mathcal{X}_1 \times \mathcal{X}_2 & n &= n_1 n_2 \\ \mathscr{L}_{\mathsf{par}} &= (\mathscr{L}_{1,\mathsf{par}} \times \mathcal{X}_2) \cup (\mathcal{X}_1 \times \mathscr{L}_{2,\mathsf{par}}) & k &= k_1 n_2 + k_2 n_1 \\ \theta_{\mathsf{par}}((C_1, C_2)) &= \theta_{1,\mathsf{par}}(C_1) \otimes \theta_{2,\mathsf{par}}(C_2) & \Gamma &= \begin{pmatrix} \Gamma^{(1)} \otimes \mathsf{Id}_{n_2} \\ \mathsf{Id}_{n_1} \otimes \Gamma^{(2)} \end{pmatrix} \end{aligned}$$

Then $\mathcal{V} = (\mathcal{X}, (\mathscr{L}_{\mathsf{par}}), \mathfrak{G}, n, \Gamma, (\theta_{\mathsf{par}}))$ is a diverse vector space corresponding to the disjunction of the two languages $\mathscr{L}_{1,\mathsf{par}}$ and $\mathscr{L}_{2,\mathsf{par}}$. A witness $\boldsymbol{\lambda}$ for a word $C = (C_1, C_2)$ is:

- $(\boldsymbol{\lambda_1} \otimes \hat{\boldsymbol{C}}_2,\ \mathbf{0} \in \mathbb{Z}_p^{n_1 k_2})$ if $\boldsymbol{\lambda_1}$ is a witness for $C_1 \in \mathscr{L}_{1,\mathsf{par}}$, with $\hat{\boldsymbol{C}}_2 = \theta_{2,\mathsf{par}}(C_2)$, or
- $(\mathbf{0} \in \mathbb{Z}_p^{k_1 n_2},\ \hat{\boldsymbol{C}}_1 \otimes \boldsymbol{\lambda_2})$ if $\boldsymbol{\lambda_2}$ is a witness for $C_2 \in \mathscr{L}_{2,\mathsf{par}}$, with $\hat{\boldsymbol{C}}_1 = \theta_{1,\mathsf{par}}(C_1)$.

Due to this, we often separate $\boldsymbol{\gamma}$ in two parts: $\boldsymbol{\gamma^{(1)}}$ corresponds to the first $k_1 n_2$ rows of $\boldsymbol{\gamma}$, while $\boldsymbol{\gamma^{(2)}}$ corresponds to the last $k_2 n_1$ rows of $\boldsymbol{\gamma}$. $\boldsymbol{\gamma^{(1)}}$ is useful to compute the hash value of (C_1, C_2) when we know a witness for C_1, while $\boldsymbol{\gamma^{(2)}}$ is useful when we know a witness for C_2.

Let us explain this construction. First, the rows of Γ generate the following subspace of $\hat{\mathcal{X}} = \mathfrak{G}^{1\times n} = \hat{\mathcal{X}}_1 \otimes \hat{\mathcal{X}}_2$:

$$\hat{\mathscr{L}} = \langle (\hat{\mathscr{L}}_1 \otimes \hat{\mathcal{X}}_2) \cup (\hat{\mathcal{X}}_1 \otimes \hat{\mathscr{L}}_2) \rangle,$$

where $\hat{\mathcal{X}}_1 = \mathfrak{G}_1^{n_1}$, $\hat{\mathcal{X}}_2 = \mathfrak{G}_2^{n_2}$, $\hat{\mathscr{L}}_1$ is the subspace of $\hat{\mathcal{X}}_1$ generated by the rows of $\Gamma^{(1)}$ and $\hat{\mathscr{L}}_2$ is the subspace of $\hat{\mathcal{X}}_2$ generated by the rows of $\Gamma^{(2)}$. So this construction corresponds exactly to the one sketched in the Section 2.1.

Then, we need to prove that $\mathcal{V}$ is really a diverse vector space, namely that $C \in \mathscr{L}_{\mathsf{par}}$ if and only if $\theta_{\mathsf{par}}(C) \in \hat{\mathscr{L}}$. Let $\hat{\boldsymbol{C}}_1 = \theta_{\mathsf{par}}(C_1)$ and $\hat{\boldsymbol{C}}_2 = \theta_{\mathsf{par}}(C_2)$. Clearly, if $C = (C_1, C_2) \in \mathscr{L}$, then $\hat{\boldsymbol{C}}_1 \in \hat{\mathscr{L}}_1$ or $\hat{\boldsymbol{C}}_2 \in \hat{\mathscr{L}}_2$ and so $\hat{\boldsymbol{C}} = \hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2 \in \hat{\mathscr{L}}$.

Now, let us prove the converse. Let $C = (C_1, C_2) \notin \mathscr{L}$. So, $\hat{\boldsymbol{C}}_1 \notin \hat{\mathscr{L}}_1$ and $\hat{\boldsymbol{C}}_2 \notin \hat{\mathscr{L}}_2$. Let H_1 and H_2 be supplementary vector spaces of $\hat{\mathscr{L}}_1$ and $\hat{\mathscr{L}}_2$ (in $\hat{\mathcal{X}}_1$ and $\hat{\mathcal{X}}_2$, respectively). Then $\hat{\mathcal{X}}_1$ is the direct sum of $\hat{\mathscr{L}}_1$ and H_1, while $\hat{\mathcal{X}}_2$ is the direct sum of $\hat{\mathscr{L}}_2$ and H_2. Therefore, $\hat{\mathscr{L}}_1 \otimes \hat{\mathcal{X}}_2$ is the direct sum of $\hat{\mathscr{L}}_1 \otimes \hat{\mathscr{L}}_2$ and $\hat{\mathscr{L}}_1 \otimes H_2$, while $\hat{\mathcal{X}}_1 \otimes \hat{\mathscr{L}}_2$ is the direct sum of $\hat{\mathscr{L}}_1 \otimes \hat{\mathscr{L}}_2$ and $H_1 \otimes \hat{\mathscr{L}}_2$. So finally, $\hat{\mathscr{L}}$ is the direct sum of $\hat{\mathscr{L}}_1 \otimes \hat{\mathscr{L}}_2$, $\hat{\mathscr{L}}_1 \otimes H_2$ and $H_1 \otimes \hat{\mathscr{L}}_2$; and $H_1 \otimes H_2$ is a supplementary of $\hat{\mathscr{L}}$. Since $0 \neq \hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2 \in H_1 \otimes H_2$, $\theta_{\mathsf{par}}(C) = \hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2 \notin \hat{\mathscr{L}}$.

Besides showing the correctness of the construction, this proof helps to better understand the structure of $\hat{\mathscr{L}}$. In particular, it shows that $\hat{\mathscr{L}}$ has dimension $l_1 l_2 + (n_1 - l_1) l_2 + l_1 (n_2 - l_2) =$

$l_1 n_2 + n_1 l_2 - l_1 l_2$, if $\hat{\mathscr{L}}_1$ has dimension l_1 and $\hat{\mathscr{L}}_2$ has dimension l_2. If the rows of $\Gamma^{(1)}$ and $\Gamma^{(2)}$ are linearly independent, $l_1 = k_1$ and $l_2 = k_2$, $\hat{\mathscr{L}}$ has dimension $k_1 n_2 + n_1 k_2 - k_1 k_2$, which is less than $k_1 n_2 + n_1 k_2$, the number of rows of Γ. Therefore the rows of Γ are never linearly independent. Actually, this last result can directly be proven by remarking that if $\hat{\boldsymbol{C}}_1 \in \hat{\mathscr{L}}_1$ and $\hat{\boldsymbol{C}}_2 \in \hat{\mathscr{L}}_2$, then $\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2 \in (\hat{\mathscr{L}}_1 \otimes \hat{\mathscr{X}}_2) \cap (\hat{\mathscr{X}}_1 \otimes \hat{\mathscr{L}}_2)$.

Concrete Equations. If we write down all equations, we get:

$$\begin{aligned}
\mathsf{hk} &= \boldsymbol{\alpha} \stackrel{\$}{\leftarrow} \mathbb{Z}_p^n \\
\mathsf{hp} &= (\gamma_j)_j \text{ with } \gamma_j = \begin{cases} \bigoplus_{l=1}^{n_1} \alpha_{(l-1)n_2+i_2} \odot \Gamma^{(1)}_{i_1,l} & \text{when } j = (i_1-1)n_2 + i_2 \\ \bigoplus_{l=1}^{n_2} \alpha_{(i_1-1)n_2+l} \odot \Gamma^{(2)}_{i_2,l} & \text{when } j = k_1 n_2 + (i_1-1)k_2 + i_2 \end{cases} \\
H &= \bigoplus_{i_1=1}^{n_1} \bigoplus_{i_2=1}^{n_2} \alpha_{i_1 n_2 + i_2} \odot \hat{C}_{1,i_1} \odot \hat{C}_{1,i_2} \\
H' &= \begin{cases} \bigoplus_{i_1=1}^{n_1} \bigoplus_{i_2=1}^{n_2} \lambda_{1,i_1} \odot \hat{C}_{2,i_2} \odot \gamma_{(i_1-1)n_2+i_2} & \text{if } \boldsymbol{\lambda_1} \odot \Gamma^{(1)} = \hat{\boldsymbol{C}}_1 \\ \bigoplus_{i_1=1}^{n_1} \bigoplus_{i_2=1}^{n_2} \lambda_{2,i_2} \odot \hat{C}_{1,i_1} \odot \gamma_{k_1 n_2 + (i_1-1)k_2 + i_2} & \text{if } \boldsymbol{\lambda_2} \odot \Gamma^{(2)} = \hat{\boldsymbol{C}}_2. \end{cases}
\end{aligned}$$

These equations are not required to understand the paper and are essentially here for the sake of completeness.

5 One-Time Simulation-Sound NIZK from Disjunctions of SPHFs

5.1 Definitions

Let us first recall some definitions from [Gro06, JR13], extended to the case of labeled non-interactive proof systems. We consider the quasi-adaptive setting of Jutla and Roy [JR13], where the common reference string (CRS) may depend on the language considered. In addition, the soundness property is only computational, so our NIZK is actually an argument and not a proof. However, this setting, though slightly weaker than usual settings, is sufficient for most cases.

Non-Interactive Proof System. Intuitively a proof system is a protocol which enables a prover to prove to a verifier that a given word or statement x is in a given NP-language. We are interested in non-interactive proofs, i.e., proofs such that the prover just sends one message.

More formally, as for SPHF, let $\mathscr{L}_{\mathsf{crs},\mathsf{par}}$ be a family of NP languages with witness relation $\mathcal{R}_{\mathsf{crs},\mathsf{par}}$, i.e., $\mathscr{L}_{\mathsf{crs},\mathsf{par}} = \{x \mid \exists w,\ \mathcal{R}_{\mathsf{crs},\mathsf{par}}(x, w) = 1\}$. We suppose that the CRS for the NIZK may depend on crs but not on par. A labeled non-interactive proof system for $(\mathscr{L}_{\mathsf{crs},\mathsf{par}})$ is defined by a tuple $\boldsymbol{\Pi} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver}, \mathcal{L})$, such that:

- $\mathcal{L}$ is a set of labels. For a classical non-interactive proof system, $\mathcal{L} = \{\perp\}$ (and in this case, the labels can be forgotten) and for a labeled one, $\mathcal{L} = \{0,1\}^*$;
- Setup is a probabilistic polynomial time algorithm which takes as input crs and outputs a common reference string (CRS) σ;
- Prove is a probabilistic polynomial time algorithm which takes as input a CRS $\sigma \stackrel{\$}{\leftarrow} \mathsf{Setup}(\mathsf{crs})$, $(\mathsf{crs}, \mathsf{par})$, a label $\ell \in \mathcal{L}$, a word $x \in \mathscr{L}$ and a witness w for x (such that $\mathcal{R}_{\mathsf{crs},\mathsf{par}}(x, w) = 1$), and outputs a proof π that x is in $\mathscr{L}$, for label ℓ;

– Ver is a deterministic algorithm which takes as input the CRS σ, crs, a label $\ell \in \mathcal{L}$, a word x and a proof π and outputs 1 to indicate acceptance and 0 otherwise;

and such that it verifies the two following properties, for any crs:

– **Perfect completeness.** A non-interactive proof is complete if an honest prover knowing a statement $x \in \mathscr{L}_{\mathsf{crs},\mathsf{par}}$ and a witness w for x can convince an honest verifier that x is in $\mathscr{L}_{\mathsf{crs},\mathsf{par}}$, for any label and any par. More formally, $\mathbf{\Pi}$ is said perfectly complete, if for all $\ell \in \mathcal{L}$, for all par, for all $x \in \mathscr{L}$ and w such that $\mathcal{R}(x, w) = 1$, for all $\sigma \overset{\$}{\leftarrow} \mathsf{Setup}(\mathsf{crs})$, we have
$$\mathsf{Ver}(\sigma, (\mathsf{crs}, \mathsf{par}), \ell, x, \mathsf{Prove}(\sigma, (\mathsf{crs}, \mathsf{par}), \ell, x, w)) = 1;$$
– **Soundness.** A non-interactive proof is said to be sound, if no polynomial time adversary $\mathcal{A}$ can prove a false statement with non-negligible probability. More formally, $\mathbf{\Pi}$ is (t, ε)-sound if for any adversary running in time at most t:
$$\Pr\Big[\sigma \overset{\$}{\leftarrow} \mathsf{Setup}(\mathsf{crs}); (\ell, x, \mathsf{par}, \pi) \overset{\$}{\leftarrow} \mathcal{A}(\sigma); \\ \mathsf{Ver}(\sigma, (\mathsf{crs}, \mathsf{par}), \ell, x, \pi) = 1 \text{ and } x \notin \mathscr{L}_{\mathsf{crs},\mathsf{par}}\Big] \leq \varepsilon.$$

Non-Interactive Zero-Knowledge Proof (NIZK). An (unbounded) NIZK (non-interative zero-knowledge proof) is a non-interactive proof system with two simulators Sim_1 and Sim_2, which can simulate Setup and Prove, but such that Sim_2 does not need any witness. More formally a NIZK is defined by a tuple $\mathbf{\Pi} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{Sim}_1, \mathsf{Sim}_2)$ such that $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ is a non-interactive proof system, and:

– Sim_1 is a probabilistic algorithm which takes as input crs and generates a CRS σ and a trapdoor $\mathcal{T}$, such that Sim_2 can use $\mathcal{T}$ to simulate proofs under σ;
– Sim_2 is a probabilistic algorithm which takes as input the CRS σ, a corresponding trapdoor σ, $(\mathsf{crs}, \mathsf{par})$, a label ℓ, a word x (not necessarily in $\mathscr{L}$), and outputs a (fake or simulated) proof π for x;

and such that it verifies the following property, for any crs:

– **Unbounded zero-knowledge** A NIZK is said (unbounded) zero-knowledge if simulated proofs are indistinguishable from real proofs. More formally, $\mathbf{\Pi}$ is (t, ε)-unbounded-zero-knowledge if, for any adversary running in time at most t:
$$\Big|\Pr\Big[\sigma \overset{\$}{\leftarrow} \mathsf{Setup}(\mathsf{crs}); \mathcal{A}(\sigma)^{\mathsf{Prove}(\sigma,(\mathsf{crs},\cdot),\cdot,\cdot,\cdot)} = 1\Big] - \\ \Pr\Big[(\sigma, \sigma) \overset{\$}{\leftarrow} \mathsf{Sim}_1(1^{\mathfrak{K}}); \mathcal{A}(\sigma)^{\mathsf{Sim}'(\sigma,\mathcal{T},(\mathsf{crs},\cdot),\cdot,\cdot,\cdot)} = 1\Big]\Big| \leq \varepsilon$$
where $\mathsf{Sim}'(\sigma, \mathcal{T}, (\mathsf{crs}, \mathsf{par}), \ell, x, w) = \mathsf{Sim}_2(\sigma, \tau, (\mathsf{crs}, \mathsf{par}), \ell, x)$, if $\mathcal{R}_{(\mathsf{crs},\mathsf{par})}(x, w) = 1$, and $\perp$ otherwise.

We are also interested in the following additional property:

– **One-Time Simulation Soundness.** A NIZK is said to be one-time simulation-sound if the adversary cannot prove a false statement, even if he can see one simulated proof for a word x of its choice. More formally, $\mathbf{\Pi}$ is (t, ε)-N-time-simulation-sound if, for any adversary running in time at most t:
$$\Pr\Big[(\sigma, \sigma) \overset{\$}{\leftarrow} \mathsf{Sim}_1(1^{\mathfrak{K}}); (x, \pi) \overset{\$}{\leftarrow} \mathcal{A}^{\mathsf{Sim}_2(\sigma,\tau,(\mathsf{crs},\cdot),\cdot,\cdot,\cdot)}(\sigma); \\ \mathsf{Ver}(\sigma, x, \pi) = 1,\ (\ell, x, \pi) \notin S \text{ and } x \notin \mathscr{L}_{\mathsf{crs},\mathsf{par}}\Big] \leq \varepsilon$$
where Sim_2 can be queried at most N times, and S is the set of (ℓ, x, π) queried to Sim_2.

5.2 NIZK from Disjunctions of SPHFs

Construction. Let us show how to construct a NIZK for any family of languages $(\mathscr{L}_{1,\mathsf{par}})_{\mathsf{par}}$ such there exists two diverse vector spaces $\mathcal{V}_1 = (\mathcal{X}_1, (\mathscr{L}_{1,\mathsf{crs}_1,\mathsf{par}}), \mathfrak{G}_1, n_1, k_1, \Gamma^{(1,\mathsf{crs}_1)}, (\theta_{1,\mathsf{crs}_1,\mathsf{par}}))$ and $\mathcal{V}_2 = (\mathcal{X}_2, (\mathscr{L}_{2,\mathsf{crs}_2}), \mathfrak{G}_2, n_2, k_2, \Gamma^{(2,\mathsf{crs}_2)}, (\theta_{2,\mathsf{crs}_2}))$ over two multiplicatively-compatible sub-graded rings $\mathfrak{G}_1$ and $\mathfrak{G}_2$ of some graded ring $\mathfrak{G}$, such that the second diverse vector space corresponds to a hard subset membership language. In particular, this construction works for any diverse vector space $\mathcal{V}_1$ where $\mathfrak{G}_1 = \mathbb{G}_1$ is a cyclic group of some bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where SXDH holds, by using as $\mathcal{V}_2$ the discrete vector space for DDH over $\mathbb{G}_2$ (Example 1).

Let $(\mathcal{X}, (\mathscr{L}_{\mathsf{par}}), \mathfrak{G}, n, \Gamma, (\theta_{\mathsf{par}}))$ be the diverse vector space corresponding to the disjunction of the two previous diverse vector spaces. Let $((\mathsf{crs}_1, \mathsf{crs}_2), (\mathcal{T}_{\mathsf{crs}_1}, \mathcal{T}_{\mathsf{crs}_2})) \stackrel{\$}{\leftarrow} \mathsf{Setup}_{\mathsf{crs}}(1^{\mathfrak{K}})$. Let $\mathsf{hk} = \boldsymbol{\alpha} \stackrel{\$}{\leftarrow} \mathbb{Z}_p^n$ be a random hashing key and $\mathsf{hp} = \boldsymbol{\gamma}$ the associated projection key. The common reference string for the NIZK is $\sigma = (\mathsf{crs}_2, \boldsymbol{\gamma})$. The trapdoor $\mathcal{T}$ is just the hashing key $\mathcal{T} = \mathsf{hk}$. Then the NIZK proof for some word $C_1 \in \mathscr{L}_{1,\mathsf{crs}_1,\mathsf{par}}$ (with witness $\boldsymbol{\lambda_1} \in \mathfrak{G}^{k_1}$) is the vector

$$\boldsymbol{\pi} := (\boldsymbol{\lambda_1} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\gamma^{(1)}} \in \mathfrak{G}_1^{n_2},$$

with $\hat{\boldsymbol{C}}_1 = \theta_{\mathsf{par}}(C_1)$ and $\boldsymbol{\gamma^{(1)}} = (\gamma_j)_{j=1,\dots,k_1 n_2}$. This can just be seen as the hash values of rows[3] of $\hat{\boldsymbol{C}}_1 \otimes \mathsf{Id}_{n_2}$.

To *verify* such a proof, we check that:

$$\Gamma^{(2)} \odot \boldsymbol{\pi} \stackrel{?}{=} (\hat{\boldsymbol{C}}_1 \otimes \mathsf{Id}_{k_2}) \odot \boldsymbol{\gamma^{(2)}}, \tag{1}$$

with $\boldsymbol{\gamma^{(2)}} = (\gamma_j)_{j=k_1 n_2+1,\dots,k_1 n_2 + n_1 k_2}$. If the proof has been generated correctly, the left hand side is equal to

$$\begin{aligned}
\Gamma^{(2)} \odot (\boldsymbol{\lambda_1} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\gamma^{(1)}} &= (\mathsf{Id}_1 \otimes \Gamma^{(2)}) \odot (\boldsymbol{\lambda_1} \otimes \mathsf{Id}_{n_2}) \odot ((\Gamma^{(1)} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\alpha}) \\
&= ((\mathsf{Id}_1 \odot \boldsymbol{\lambda_1}) \otimes (\Gamma^{(2)} \odot \mathsf{Id}_{n_2})) \odot (\Gamma^{(1)} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\alpha} \\
&= (\boldsymbol{\lambda_1} \otimes \Gamma^{(2)}) \odot (\Gamma^{(1)} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\alpha} \\
&= ((\boldsymbol{\lambda_1} \odot \Gamma^{(1)}) \otimes (\Gamma^{(2)} \odot \mathsf{Id}_{n_2})) \odot \boldsymbol{\alpha},
\end{aligned}$$

while the right hand side is always equal to:

$$(\hat{\boldsymbol{C}}_1 \otimes \mathsf{Id}_{k_2}) \odot \boldsymbol{\gamma^{(2)}} = (\hat{\boldsymbol{C}}_1 \otimes \mathsf{Id}_{k_2}) \odot (\mathsf{Id}_{n_1} \otimes \Gamma^{(2)}) \odot \boldsymbol{\alpha} = ((\hat{\boldsymbol{C}}_1 \odot \mathsf{Id}_{n_1}) \otimes (\mathsf{Id}_{k_2} \odot \Gamma^{(2)})) \odot \boldsymbol{\alpha},$$

which is the same as the left hand side, since $\boldsymbol{\lambda_1} \odot \Gamma^{(1)} = \hat{\boldsymbol{C}}_1 \odot \mathsf{Id}_{n_1}$ and $\Gamma^{(2)} \odot \mathsf{Id}_{n_2} = \mathsf{Id}_{k_2} \odot \Gamma^{(2)}$. Hence the *completeness*. Another way to see it, is that the row i_2 of the right hand side is the hash value of "$(\hat{\boldsymbol{C}}_1, \boldsymbol{e_{2,i_2}} \odot \Gamma^{(2)})$" computed using the witness $\boldsymbol{\lambda_2} = \boldsymbol{e_{2,i_2}}$, while the row i_2 of the left hand side is this hash value computed using the witness $\boldsymbol{\lambda_1}$.

To simulate a proof $\boldsymbol{\pi}$ for C_1, we just compute it as follows:

$$\boldsymbol{\pi} := (\hat{\boldsymbol{C}}_1 \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\alpha}.$$

The (perfect) unbounded *zero-knowledge* property comes from the fact that the normal proof for $C_1 \in \mathscr{L}_1$ with witness $\boldsymbol{\lambda_1}$ is:

$$\boldsymbol{\pi} = (\boldsymbol{\lambda_1} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\gamma^{(1)}} = (\boldsymbol{\lambda_1} \otimes \mathsf{Id}_{n_2}) \odot (\Gamma^{(1)} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\alpha} = ((\boldsymbol{\lambda_1} \odot \Gamma^{(1)}) \otimes (\mathsf{Id}_{n_2} \odot \mathsf{Id}_{n_2})) \odot \boldsymbol{\alpha},$$

which is equal to the simulated proof for C_1, since $\hat{\boldsymbol{C}}_1 = \boldsymbol{\lambda_1} \odot \Gamma^{(1)}$ and $\mathsf{Id}_{n_2} \odot \mathsf{Id}_{n_2} = \mathsf{Id}_{n_2}$.

[3] This is not quite accurate, since rows of $\hat{\boldsymbol{C}}_1 \otimes \mathsf{Id}_{n_1}$ are not words in $\mathcal{X}$ but in $\hat{\mathcal{X}}$. But to give intuition, we will often make this abuse of notation.

Soundness Proof. It remains to prove the soundness property, under the hard subset membership of $\mathscr{L}_2$. We just need to show that if the adversary is able to generate a valid proof $\boldsymbol{\pi}$ for a word $C_1 \notin \mathscr{L}_1$, then we can use $\boldsymbol{\pi}$ to check if a word C_2 is in $\mathscr{L}_2$ or not. More precisely, let $C_2 \in \mathcal{X}_2$, let H be the hash value of (C_1, C_2) computed using hk, and let H' be the following value:

$$H' \coloneqq \hat{\boldsymbol{C}}_2 \odot \boldsymbol{\pi}.$$

On the one hand, if $C_2 \in \mathscr{L}_2$, there exists a witness $\boldsymbol{\lambda_2}$ such that $\hat{\boldsymbol{C}}_2 = \boldsymbol{\lambda_2} \odot \Gamma^{(2)}$ and so, thanks to Equation (1):

$$H' = \boldsymbol{\lambda_2} \odot \Gamma^{(2)} \odot \boldsymbol{\pi} = \boldsymbol{\lambda_2} \odot (\hat{\boldsymbol{C}}_1 \otimes \mathsf{Id}_{k_2}) \odot \boldsymbol{\gamma^{(2)}} = (\hat{\boldsymbol{C}}_1 \otimes \boldsymbol{\lambda_2}) \odot \boldsymbol{\gamma^{(2)}} = H,$$

the last equality coming from the correctness of the SPHF and the fact the last-but-one expression is just the hash value of (C_1, C_2) computed using $\mathsf{ProjHash}$ and witness $\boldsymbol{\lambda_2}$.

On the other hand, if $C_2 \notin \mathscr{L}_2$, then $(C_1, C_2) \notin \mathscr{L}$. So H looks completely random by smoothness and the probability that $H' = H$ is at most $1/|\Pi|$.

Toward One-Time Simulation Soundness. The previous proof does not work anymore if the adversary is allowed to get even one single simulated proof of a word $C_1 \notin \mathscr{L}_1$. Indeed, in this case, the smoothness does not hold anymore, in the above proof of soundness. That is why we need a stronger form of smoothness for SPHF, called 2-smoothness.

5.3 2-Smooth Projective Hash Functions

Definition. In order to define the notion of 2-smoothness, let us first introduce the notion of tag-SPHF. A *tag-SPHF* is similar to an SPHF except that Hash and $\mathsf{ProjHash}$ now take a new input, called a tag $\mathsf{tag} \in \mathsf{Tags}$. Similarly a *tag diverse vector space* is a diverse vector space where the function θ also takes as input a tag $\mathsf{tag} \in \mathbb{Z}_p$. The vector $\boldsymbol{\lambda}$ is now allowed to depend on tag, but the matrix Γ is independent of tag.

A 2-smooth SPHF is a tag-SPHF for which the hash value of a word $C \in \mathcal{X}$ for a tag tag looks random even if we have access to the hash value of another word $C' \in \mathcal{X}$ for a different tag $\mathsf{tag}' \neq \mathsf{tag}$. Formally, an SPHF is ε-2-smooth if for any crs, for all functions f' and f from the set of projection keys to $\mathsf{Params}_{\mathsf{crs}} \times \mathcal{X} \times \mathsf{Tags}$, so that $(\mathsf{par}', C', \mathsf{tag}') = f'(\mathsf{hp})$ and $(\mathsf{par}, C, \mathsf{tag}) = f(\mathsf{hp})$ are such that $C \notin \mathscr{L}_{\mathsf{crs},\mathsf{par}}$ and $\mathsf{tag}' \neq \mathsf{tag}$, the two following distributions are ε-statistically close:

$$\left\{ (\mathsf{hp}, H', H) \;\middle|\; \begin{array}{ll} \mathsf{hk} \xleftarrow{\$} \mathsf{HashKG}(\mathsf{crs}); & \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{crs}); \\ (\mathsf{par}', C', \mathsf{tag}') \leftarrow f'(\mathsf{hp}) & H' \leftarrow \mathsf{Hash}(\mathsf{hk}, (\mathsf{crs}, \mathsf{par}'), C', \mathsf{tag}'); \\ (\mathsf{par}, C, \mathsf{tag}) \leftarrow f(\mathsf{hp}); & H \leftarrow \mathsf{Hash}(\mathsf{hk}, (\mathsf{crs}, \mathsf{par}), C, \mathsf{tag}) \end{array} \right\}$$

$$\left\{ (\mathsf{hp}, H', H) \;\middle|\; \begin{array}{ll} \mathsf{hk} \xleftarrow{\$} \mathsf{HashKG}(\mathsf{crs}); & \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{crs}); \\ (\mathsf{par}', C', \mathsf{tag}') \leftarrow f'(\mathsf{hp}); & H' \leftarrow \mathsf{Hash}(\mathsf{hk}, (\mathsf{crs}, \mathsf{par}'), C', \mathsf{tag}'); \\ (\mathsf{par}, C, \mathsf{tag}) \leftarrow f(\mathsf{hp}); & H \xleftarrow{\$} \Pi \end{array} \right\}$$

In practice the tag tag_C of a word C will often be its hash value from a collision-resistant hash function.

The 2-smoothness is very similar to the 2-universality of Cramer and Shoup in [CS02]. There are however two minor differences, the first being the existence of an explicit tag, and second being the fact that the hash value of a word outside the language is supposed to be uniformly random instead of just having some entropy. This slightly simplifies its usage in our constructions, in our opinion.

Canonical Construction from Diverse Vector Spaces. Let $\mathcal{V} = (\mathcal{X}, (\mathscr{L}_{\mathsf{par}})_{\mathsf{par}}, \mathfrak{G}, n, k, \Gamma, (\theta_{\mathsf{par}})_{\mathsf{par}})$ be a diverse vector space. If we set:

$$(\tilde{n}, \tilde{k}) = (2n, 2k) \quad \tilde{\Gamma} = \begin{pmatrix} \Gamma & 0 \\ 0 & \Gamma \end{pmatrix} \quad \tilde{\theta}_{\mathsf{par}}(C, \mathsf{tag}) = (\hat{\boldsymbol{C}},\, \mathsf{tag} \odot \hat{\boldsymbol{C}}) = (\theta_{\mathsf{par}}(C),\, \mathsf{tag} \odot \theta_{\mathsf{par}}(C)),$$

then $\tilde{\mathcal{V}} = (\mathcal{X}, (\mathscr{L}_{\mathsf{par}}), \mathfrak{G}, \tilde{n}, \tilde{k}, \tilde{\Gamma}, (\tilde{\theta}_{\mathsf{par}}))$ is a 2-smooth diverse vector space.

The vector $\tilde{\boldsymbol{\lambda}}$ for a word $C \in \mathscr{L}_{\mathsf{par}}$ and a tag tag is just

$$\tilde{\boldsymbol{\lambda}} = (\boldsymbol{\lambda},\, \mathsf{tag} \odot \boldsymbol{\lambda}),$$

and it is clear that $C \in \mathscr{L}_{\mathsf{par}}$ if and only if $\tilde{\hat{\boldsymbol{C}}} = \tilde{\theta}_{\mathsf{par}}(C, \mathsf{tag})$ is a linear combination of rows of Γ.

Let us now prove the 2-smoothness. Let $C' \in \mathcal{X}$ and $C \in \mathcal{X} \setminus \mathscr{L}_{\mathsf{par}}$, and tag' and tag be two distinct tags. We have

$$\tilde{\hat{\boldsymbol{C}}}' = (\hat{\boldsymbol{C}}',\, \mathsf{tag}' \odot \hat{\boldsymbol{C}}') \qquad \text{and} \qquad \tilde{\hat{\boldsymbol{C}}} = (\hat{\boldsymbol{C}},\, \mathsf{tag} \odot \hat{\boldsymbol{C}}).$$

We just need to prove that $\tilde{\hat{\boldsymbol{C}}}$ is not in the subspace generated by the rows of Γ and $\tilde{\hat{\boldsymbol{C}}}'$, or in other words that it is not in $\hat{\mathscr{L}}' = \langle \hat{\mathscr{L}} \cup \{\tilde{\hat{\boldsymbol{C}}}'\} \rangle$. Indeed, in that case, H' could just be seen as a part of the projection key for the language $\hat{\mathscr{L}}'$, and by smoothness, we get that H looks uniformly random.

So it remains to prove that linear independence of $\tilde{\hat{\boldsymbol{C}}}$. By contraposition let us suppose there exists $\tilde{\boldsymbol{\lambda}} \in \mathbb{Z}_p^{2k}$ and μ such that:

$$\tilde{\hat{\boldsymbol{C}}} = (\hat{\boldsymbol{C}},\, \mathsf{tag} \odot \hat{\boldsymbol{C}}) = \tilde{\boldsymbol{\lambda}} \odot \Gamma \oplus \mu \odot \tilde{\hat{\boldsymbol{C}}}' = \tilde{\boldsymbol{\lambda}} \odot \begin{pmatrix} \Gamma & 0 \\ 0 & \Gamma \end{pmatrix} \oplus \mu \odot (\hat{\boldsymbol{C}}',\, \mathsf{tag}' \odot \hat{\boldsymbol{C}}').$$

Therefore $\tilde{\hat{\boldsymbol{C}}} \ominus \mu \odot \tilde{\hat{\boldsymbol{C}}}'$ and $\mathsf{tag} \odot \tilde{\hat{\boldsymbol{C}}} \ominus \mathsf{tag}' \odot \mu \odot \tilde{\hat{\boldsymbol{C}}}'$ are both linear combination of rows of Γ, and so is

$$\mathsf{tag}' \odot (\tilde{\hat{\boldsymbol{C}}} \ominus \mu \odot \tilde{\hat{\boldsymbol{C}}}') \;\ominus\; (\mathsf{tag} \odot \tilde{\hat{\boldsymbol{C}}} \ominus \mathsf{tag}' \odot \mu \odot \tilde{\hat{\boldsymbol{C}}}') \;=\; (\mathsf{tag}' - \mathsf{tag}) \odot \tilde{\hat{\boldsymbol{C}}}.$$

Since $\mathsf{tag}' - \mathsf{tag} \neq 0$, this implies that $\tilde{\hat{\boldsymbol{C}}}$ is also a linear combination of rows of Γ, hence $C \in \mathscr{L}_{\mathsf{par}}$, which is not the case.

5.4 One-Time Simulation-Sound Zero-Knowledge Arguments from SPHF

Let us now replace the first diverse vector space by its canonical 2-smooth version in the NIZK construction of Section 5.2. The resulting construction is a one-time simulation-sound NIZK, if $\hat{\boldsymbol{C}}_1$ is computed as $\theta_{1,\mathsf{par}}(C_1, \mathsf{tag})$ where tag is the hash value of (C_1, ℓ) under some collision-resistant hash function $\mathcal{H}$: $\mathsf{tag} = \mathcal{H}((C_1, \ell))$.

Completeness and perfect zero-knowledge can be proven the same way. It remains to prove the one-time simulation soundness. The proof is similar to the one in Section 5.2, except for the final step: proving that the hash value H of (C_1, C_2) with tag $\mathsf{tag} = \mathcal{H}((C_1, \ell))$ looks random even if the adversary sees a simulated NIZK π' for a word $C_1' \in \mathcal{X}_1$ and label ℓ'.

We first remark that the tag tag' can be supposed distinct from the tag tag for the NIZK π created by the adversary, thanks to the collision-resistance of $\mathcal{H}$. We recall that π' is the hash values of the rows of $\hat{\boldsymbol{C}}_1' \otimes \mathsf{Id}_{n_2}$. So to prove that the hash value of (C_1, C_2) with tag tag looks random even with access to π', we just need to remark that $\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2$ is linearly independent of rows of Γ and $\hat{\boldsymbol{C}}_1' \otimes \mathsf{Id}_{n_2}$. The proof is similar to the proof of 2-smoothness.

Remark 7. It would be easy to extend this construction to handle N-time simulation-sound NIZK, for any constant N. The NIZK CRS σ size would just be N times larger compared to the NIZK construction of Section 5.2, and the proof size would remain constant.

5.5 Concrete Instantiation

If $\mathcal{V}_1$ is a diverse vector space over $\mathbb{G}_1$ and $\mathcal{V}_2$ is the diverse vector space for DDH in $\mathbb{G}_2$, where $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ is a bilinear group where DDH is hard in $\mathbb{G}_2$, then we get a NIZK and a one-time simulation sound NIZK whose proof is composed of only $n_2 = 2$ group elements in $\mathbb{G}_1$.

More generally, we can use as $\mathcal{V}_2$, the following diverse vector space for the language corresponding to κ-Lin: $\mathcal{X}_2 = \hat{\mathcal{X}}_2 = \mathbb{G}_2^{\kappa+1}$, $\mathsf{crs}_2 = (g_2, \zeta_1, \dots, \zeta_\kappa) \xleftarrow{\$} \mathbb{G}_2^{\kappa+1}$, θ_2 is the identity function and $\hat{\mathscr{L}}_2 = \mathscr{L}_2$ is defined by the following matrix:

$$\Gamma_2 = \begin{pmatrix} \zeta_1 & 1 & \dots & 1 & g_2 \\ 1 & \zeta_2 & \dots & 1 & g_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \dots & \zeta_\kappa & g_2 \end{pmatrix},$$

where 1 is the element $g_2^0 \in \mathbb{G}_2$. A word $C_2 = \hat{\boldsymbol{C}}_2 = (\hat{C}_{2,1}, \dots, \hat{C}_{2,\kappa+1}) \in \mathcal{X}_2$ is in $\mathscr{L}_2$ if and only if $(\zeta_1, \dots, \zeta_\kappa, \hat{C}_{2,1}, \dots, \hat{C}_{2,\kappa}, \hat{C}_{2,\kappa+1})$ is a κ-Lin-tuple (see Section 3.4). This yields a proof consisting of only $n_2 = \kappa + 1$ group elements, under κ-Lin. The DDH case corresponds to $\kappa = 1$.

Languages handled are exactly languages for which there exists such a diverse vector space $\mathcal{V}_1$ over $\mathbb{G}_1$. That corresponds to languages handled by Jutla and Roy NIZK [JR13], which they call linear subspaces (assuming θ is the identity function), if we forget the fact that in [JR13], it is supposed that crs can be generated in such a way that discrete logarithms of Γ is known (that is what they call *witness-samplable* languages). That encompasses DDH, κ-Lin, and languages of ElGamal, Cramer-Shoup or similar ciphertexts whose plaintexts verify some linear system of equations, as already shown in [BBC+13].

Concrete comparison with existing work can be found in Section 8.3.

5.6 Application: Threshold Cramer-Shoup-like Encryption Scheme

The Cramer-Shoup public-key encryption scheme [CS98] is one of the most efficient $\mathsf{IND\text{-}CCA}$ encryption schemes with a proof of security in the standard model. We remark here that, if we replace the last part of a Cramer-Shoup ciphertext (the 2-universal projective hash proof or w in our notations in Section A.3) by a one-time simulation-sound NIZK on the DDH language, we can obtain an $\mathsf{IND\text{-}CCA}$ scheme supporting threshold decryption. Intuitively, this comes from the fact that the resulting scheme becomes "publicly verifiable", in the sense that, after verifying the NIZK (which is publicly verifiable), one can obtain the underlying message via "simple" algebraic operations which can easily be "distributed".

Previous one-time simulation-sound NIZK were quite inefficient and the resulting scheme would have been very inefficient compared to direct constructions of threshold $\mathsf{IND\text{-}CCA}$ encryption schemes. However, in our case, our new one-time simulation-sound NIZK based on disjunction of SPHF only adds one group element to the ciphertext (compared to original Cramer-Shoup encryption scheme). A detailed comparison is given in Section 8.4, where we also introduce a more efficient version of that threshold encryption scheme, for which the ciphertexts have the same size as the ciphertexts of the original Cramer-Shoup encryption scheme.

Here is an explicit description of the scheme (details and explanations can be found in Appendix C.1):

- $\mathsf{Setupe}(1^{\mathfrak{K}})$ generates the following parameters param: an asymmetric bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$;
- $\mathsf{KGe}(\mathsf{param})$ chooses two random generators $g_{1,1}$ and $g_{1,2}$ of $\mathbb{G}_1$, and a random scalar $z \xleftarrow{\$} \mathbb{Z}_p$ and sets $h \leftarrow g_{1,1}^z$. It also chooses a random generator h_2 of $\mathbb{G}_2$. Then, it chooses a random

vector $\boldsymbol{\alpha} \stackrel{\$}{\leftarrow} \mathbb{Z}_p^8$ and computes the following two vectors

$$\boldsymbol{\gamma}^{\mathbf{(1)}} = (\gamma_1^{(1)}, \gamma_2^{(1)}, \gamma_3^{(1)}, \gamma_4^{(1)}) \leftarrow (g_{1,1}^{\alpha_1} \cdot g_{1,2}^{\alpha_3},\ g_{1,1}^{\alpha_2} \cdot g_{1,2}^{\alpha_4},\ g_{1,1}^{\alpha_5} \cdot g_{1,2}^{\alpha_7},\ g_{1,1}^{\alpha_6} \cdot g_{1,2}^{\alpha_8}),$$
$$\boldsymbol{\gamma}^{\mathbf{(2)}} = (\gamma_1^{(2)}, \gamma_2^{(2)}, \gamma_3^{(2)}, \gamma_4^{(2)}) \leftarrow (g_2^{\alpha_1} \cdot h_2^{\alpha_2},\ g_2^{\alpha_3} \cdot h_2^{\alpha_4},\ g_2^{\alpha_5} \cdot h_2^{\alpha_6},\ g_2^{\alpha_7} \cdot h_2^{\alpha_2}).$$

These two vectors together with g_2 and h_2 correspond to the NIZK CRS for the following language:

$$\mathcal{R}_{1,\mathsf{crs}_1}((u_1, u_2), r) = 1 \qquad \text{if and only if} \qquad u_1 = g_{1,1}^r \quad \text{and} \quad u_2 = g_{1,2}^r.$$

Finally, it chooses a hash function $\mathcal{H}$ in a collision-resistant hash function family $\mathcal{HF}$ and outputs the encryption key (or public key) $\mathsf{ek} = (g_{1,1}, g_{1,2}, h, \boldsymbol{\gamma}^{\mathbf{(1)}}, \mathcal{H})$ and the decryption key (or secret key) $\mathsf{dk} = (z, g_2, h_2, \boldsymbol{\gamma}^{\mathbf{(2)}}, \mathcal{H})$.

– $\mathsf{Encrypt}(\ell, \mathsf{ek}, M; r)$, for a message $M \in \mathbb{G}$ and a random scalar $r \in \mathbb{Z}_p$, outputs the following ciphertext

$$C = \left(\ell,\ u_1 = g_{1,1}^r,\ u_2 = g_{1,2}^r,\ v = M \cdot h^r,\ \pi_1 = (\gamma_1^{(1)})^r \cdot (\gamma_3^{(1)})^{r\xi},\ \pi_2 = (\gamma_2^{(1)})^r \cdot (\gamma_4^{(1)})^{r\xi}\right),$$

where $\xi = \mathcal{H}((\ell, u_1, u_2, v))$. The vector $\boldsymbol{\pi} = (\pi_1, \pi_2)$ is the proof that the ciphertext if valid.

– $\mathsf{Decrypt}(\ell, \mathsf{dk}, C)$ first computes $\xi = \mathcal{H}((\ell, u_1, u_2, v))$ and checks the following equality:

$$e(\pi_1, g_2) \cdot e(\pi_2, h_2) \stackrel{?}{=} e(u_1, \gamma_1^{(2)} \cdot (\gamma_3^{(2)})^{\xi}) \cdot e(u_2, \gamma_2^{(2)} \cdot (\gamma_4^{(2)})^{\xi}), \tag{2}$$

which corresponds to the verification of the NIZK proof $\boldsymbol{\pi} = (\pi_1, \pi_2)$. If the equality holds, it computes $M = v/u_1^z$ and outputs M. Otherwise, it outputs $\perp$.

We remark that the decryption consists in a first equality test (Equation (2)) which can be performed knowing only g_2, h_2 and $\boldsymbol{\gamma}^{\mathbf{(2)}}$, followed (if the test passes) by a computation similar to an ElGamal decryption ($M = v/u_1^z$). By secret sharing z, this last operation can be performed in a threshold way, while the equality test can be performed publicly, if the values g_2, h_2 and $\boldsymbol{\gamma}^{\mathbf{(2)}}$ are made public (which does not hinder IND-CCA security). This justifies our claim that this scheme is a threshold encryption scheme.

In addition, as shown in Appendix C.1 the decryption can be performed without computing pairing, but only exponentiations (as the original Cramer-Shoup encryption scheme), by replacing the original equality test (Equation (2)) by the following equality tests:

$$\pi_1 \stackrel{?}{=} u_1^{\alpha_1 + \xi\alpha_5} \cdot u_2^{\alpha_3 + \xi\alpha_7} \qquad\qquad \pi_2 \stackrel{?}{=} u_1^{\alpha_2 + \xi\alpha_6} \cdot u_2^{\alpha_4 + \xi\alpha_8},$$

assuming $\boldsymbol{\alpha}$ is part of the decryption key (in which case g_2, h_2, and $\boldsymbol{\gamma}^{\mathbf{(2)}}$ need not to be part of dk). Each of this new equality test is very similar to the equality test of the original Cramer-Shoup (on the last part of the ciphertext, with 2-universal hash proof systems). This justifies our claim that our scheme is a threshold variant of Cramer-Shoup encryption scheme.

6 Other Applications of Disjunctions of SPHF

6.1 One-Round Group Password Authenticated Key Exchange

Definition. A one-round group password authenticated key exchange (GPAKE) is a protocol enabling n users sharing a password pw to establish a common secret key sk in only one round: just by sending one flow. The formal security model of Abdalla et al. [ABCP06] is recalled in Appendix C.2. Basically, we want that an adversary controlling $q_{\mathtt{active}}$ users cannot win with probability more than $q_{\mathtt{active}}/N$, where N is the number of possible passwords. Winning means distinguishing a real key (generated by an honest user following the protocol) from a random key sk.

Our Construction. Let us now show the first one-round group password authenticated key exchange (in the common reference string model) for more than 2 users. It is inspired by the one-round PAKE of Katz and Vaikuntanathan [KV11]. Basic ideas of this protocol can be found in the end of Section 2.

We only describe the protocol for 3 players for the sake of simplicity. But it can easily be extended to n players using $(n-1)$-way symmetric multilinear maps. However, since each player needs to send an exponential number of group elements in n, this protocol is limited to small number n of users. Notice that the only known group one-round (non authenticated) key exchange is the group Diffie-Hellman key exchange [BS03], which also require $(n-1)$-way symmetric multilinear maps.

In our protocol, we suppose that users participating in our one-round GPAKE are identified as U_1, U_2 and U_3. Let $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ be a symmetric bilinear group. The common reference string crs contains an encryption key ek for the linear Cramer-Shoup encryption scheme. The linear Cramer-Shoup encryption scheme [Sha07] recalled in Appendix A.2 is a variant of the Cramer-Shoup encryption scheme secure under DLin. We cannot use the original Cramer-Shoup scheme because DDH does not hold in symmetric bilinear groups. Let $\mathscr{L}_{\mathsf{crs},\mathsf{par}}$ be the language of tuples $((\ell_1, C_1), (\ell_2, C_2))$, where C_1 (with label ℓ_1) or C_2 (with label ℓ_2) is a valid linear Cramer-Shoup ciphertext of par. This language is the disjunction of two identical languages $\mathscr{L}_{1,\mathsf{par}} = \mathscr{L}_{2,\mathsf{par}}$, namely the language of the valid linear Cramer-Shoup ciphertexts of par. A diverse vector space for $\mathscr{L}_{1,\mathsf{par}}$ is recalled in Appendix C.2.

Let us describe the protocol for U_1 with password pw_1 (the protocol is symmetric for the other users): user U_1 first generates an hashing key hk_1 and an associated projection key hp_1 for a 2-smooth SPHF on $\mathscr{L}_{\mathsf{par}}$ with $\mathsf{par} = \mathsf{pw}_1$. Such a 2-smooth SPHF can be created using the generic transformation of Section 5.3. Then, he generates $C_1 \stackrel{\$}{\leftarrow} \mathsf{Encrypt}(\ell, \mathsf{ek}, \mathsf{pw}_1; r_1, s_1)$, with label $\ell = (U_1, U_2, U_3, \mathsf{hp}_1)$ and random scalars $r_1, s_1 \stackrel{\$}{\leftarrow} \mathbb{Z}_p$. Finally, he sends C_1, hp_1 to U_2 and U_3.

Then after receiving C_2, hp_2 from U_2 and C_3, hp_3 from U_3, U_1 computes sk as the product ($\cdot$ in $\mathbb{G}_T$) of the three following values:

$$\begin{aligned}
&\mathsf{Hash}(\mathsf{hk}_1, (\mathsf{crs}, \mathsf{pw}_1), ((\ell_2, C_2), (\ell_3, C_3)))\\
&\mathsf{ProjHash}(\mathsf{hp}_2, (\mathsf{crs}, \mathsf{pw}_1), ((\ell_1, C_1), (\ell_3, C_3)), (r_1, s_1))\\
&\mathsf{ProjHash}(\mathsf{hp}_3, (\mathsf{crs}, \mathsf{pw}_1), ((\ell_1, C_1), (\ell_2, C_2)), (r_1, s_1)),
\end{aligned}$$

where $\ell_2 = (U_1, U_2, U_3, \mathsf{hp}_2)$ and $\ell_3 = (U_1, U_2, U_3, \mathsf{hp}_3)$. Concretely, each user sends a ciphertext with 5 group elements, and a projection key with $2 \times 7 \times 4 = 56$ group elements, which adds up to 61 group elements sent per user.

This protocol is secure under DLin, as proven in Appendix C.2. The proof is a very delicate extension of the proof of the one-round PAKE of Katz and Vaikuntanathan in [KV11], and may be of independent interest.

Its extension to n users would require to use an n-linear variant of Cramer-Shoup and to use an $(n-1)$-smooth SPHF. The proof would hold under n-Lin. However, the size of the projection keys and the gap in the security reduction grow exponentially in n, and so we are limited to small values of n, which needs to be logarithmic in the security parameter $\mathfrak{K}$.

6.2 Trapdoor Smooth Projective Hash Functions

Definition. A TSPHF [BBC+13] is an extension of a classical SPHF with an additional algorithm TSetup, which takes as input the CRS crs and outputs an additional CRS crs' and a trapdoor $\mathcal{T}_{\mathsf{crs}'}$ specific to crs', which can be used to compute the hash value of words C knowing only hp. The additional CRS crs' is often implicit.

TSPHFs enable to construct efficient PAKE protocols in the UC model and also efficient 2-round zero-knowledge proofs. For the latter, the trapdoor is used to enable the simulator to simulate a prover playing against a dishonest verifier.

Formally, a TSPHF is defined by seven algorithms:

- $\mathsf{TSetup}(\mathsf{crs})$ takes as input the CRS crs (generated by $\mathsf{Setup}_{\mathsf{crs}}$) and generates the second CRS crs', together with a trapdoor $\mathcal{T}_{\mathsf{crs}'}$;
- HashKG, ProjKG, Hash, and $\mathsf{ProjHash}$ behave as for a classical SPHF;
- $\mathsf{VerHP}(\mathsf{hp}, \mathsf{crs})$ outputs 1 if hp is a valid projection key, and 0 otherwise.
- $\mathsf{THash}(\mathsf{hp}, (\mathsf{crs}, \mathsf{par}), C, \mathcal{T}_{\mathsf{crs}'})$ outputs the hash value of C from the projection key hp and the trapdoor $\mathcal{T}_{\mathsf{crs}'}$.

It must verify the following properties:

- *Correctness* is defined by two properties: *hash correctness*, which corresponds to correctness for classical SPHFs, and an additional property called *trapdoor correctness*, which states that, for any $C \in \mathcal{X}$, if hk and hp are honestly generated, we have: $\mathsf{VerHP}(\mathsf{hp}, \mathsf{crs}) = 1$ and $\mathsf{Hash}(\mathsf{hk}, (\mathsf{crs}, \mathsf{par}), C) = \mathsf{THash}(\mathsf{hp}, (\mathsf{crs}, \mathsf{par}), C, \mathcal{T}_{\mathsf{crs}'})$, with overwhelming probability;
- *Smoothness* cannot obviously be statistical because of THash. So smoothness is defined by the experiments $\mathsf{Exp}^{\mathtt{smooth}\text{-}b}$ depicted in Fig. 2. We suppose that testing $C \in \mathscr{L}_{\mathsf{crs},\mathsf{par}}$ can be done in polynomial-time using $\mathcal{T}_{\mathsf{crs}}$.

$\mathsf{Exp}^{\mathtt{smooth}\text{-}b}(\mathcal{A}, \mathfrak{K})$
    $(\mathsf{crs}, \mathcal{T}_{\mathsf{crs}}) \xleftarrow{\$} \mathsf{Setup}_{\mathsf{crs}}(1^{\mathfrak{K}})$
    $\mathsf{hk} \xleftarrow{\$} \mathsf{HashKG}(\mathsf{crs})$
    $\mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{crs})$
    $(\mathsf{par}, C, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(\mathsf{crs})$
    if $b = 0$ or $C \in \mathscr{L}_{\mathsf{crs},\mathsf{par}}$ **then**
        $H \leftarrow \mathsf{Hash}(\mathsf{hk}, (\mathsf{crs}, \mathsf{par}), C)$
    else
        $H \xleftarrow{\$} \Pi$
    return $\mathcal{A}(\mathsf{st}, H)$

Fig. 2. Experiments $\mathsf{Exp}^{\mathtt{smooth}\text{-}b}$ for computational smoothness of TSPHF

- The (t, ε)-*soundness* property says that, given crs, $\mathcal{T}_{\mathsf{crs}}$ and crs', no adversary running in time at most t can produce a projection key hp, a value par, a word C and valid witness w such that hp is valid (i.e., $\mathsf{VerHP}(\mathsf{hp}, (\mathsf{crs}, \mathsf{par})) = 1$) but

  $$\mathsf{THash}(\mathsf{hp}, (\mathsf{crs}, \mathsf{par}), C, \mathcal{T}_{\mathsf{crs}'}) \neq \mathsf{ProjHash}(\mathsf{hp}, (\mathsf{crs}, \mathsf{par}), C, w),$$

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

It is important to notice that $\mathcal{T}_{\mathsf{crs}}$ is not an input of THash and it is possible to use THash, while generating crs with an algorithm which cannot output $\mathcal{T}_{\mathsf{crs}}$ (as soon as the distribution of crs output by this algorithm is indistinguishable from the one output by $\mathsf{Setup}_{\mathsf{crs}}$, obviously). For example, if $\mathcal{T}_{\mathsf{crs}}$ contains a decryption key, it is still possible to use the $\mathsf{IND\text{-}CPA}$ game for the encryption scheme, while making calls to THash. $\mathcal{T}_{\mathsf{crs}}$ is just used in the definition of the computational smoothness and in the proof of this property.

New Construction. Let us now show how to construct a TSPHF for any family of languages $(\mathscr{L}_{1,\mathsf{par}})_{\mathsf{par}}$ such that there exists two diverse vector spaces $\mathcal{V}_1 = (\mathcal{X}_1, (\mathscr{L}_{1,\mathsf{crs}_1,\mathsf{par}}), \mathfrak{G}_1, n_1, k_1,$

$\Gamma^{(1,\mathsf{crs}_1)}, (\theta_{1,\mathsf{crs}_1,\mathsf{par}}))$ and $\mathcal{V}_2 = (\mathcal{X}_2, (\mathscr{L}_{2,\mathsf{crs}_2}), \mathfrak{G}_2, n_2, k_2, \Gamma^{(2,\mathsf{crs}_2)}, (\theta_{2,\mathsf{crs}_2}))$ over two multiplicatively-compatible sub-graded ring $\mathfrak{G}_1$ and $\mathfrak{G}_2$ of some graded ring $\mathfrak{G}$, such that the second diverse vector space corresponds to a hard subset membership language.

This is actually exactly the same requirement as for NIZK from SPHFs in Section 5.2.

Let $\mathcal{V} = (\mathcal{X}, (\mathscr{L}_{\mathsf{par}}), \mathfrak{G}, n, \Gamma, (\theta_{\mathsf{par}}))$ be the diverse vector space corresponding to the disjunction of the two previous diverse vector spaces. Let crs' contains a random word $C_2 \stackrel{\$}{\leftarrow} \mathscr{L}_2$ and $\mathcal{T}_{\mathsf{crs}'} = \boldsymbol{\lambda_2}$ be its witness. Then, the algorithms HashKG and ProjKG are the same as the one for $\mathcal{V}$, while the hash value of a word $C_1 \in \mathcal{X}_1$ is just the hash value of (C_1, C_2). This can be computed in three ways:

- Hash: using hk
- ProjHash: using a witness $\boldsymbol{\lambda_1}$ for $C_1 \in \mathscr{L}_1$ (if $C_1 \in \mathscr{L}_1$)
- THash: using the witness $\mathcal{T}_{\mathsf{crs}'} = \boldsymbol{\lambda_2}$ for $C_2 \in \mathscr{L}_2$.

The correctness is trivial, and the computational smoothness directly comes from the smoothness of $\mathcal{V}$ and the hard subset membership problem for $\mathscr{L}_2$: $\mathsf{crs}' = C_2$ is indistinguishable from a word $\mathsf{crs}' = C_2 \notin \mathscr{L}_2$, and in this case, the hash value of (C_1, C_2) is statistically indistinguishable from random, when $C_1 \notin \mathscr{L}_1$.

It remains to define correctly VerHP to get the perfect soundness property. For that, VerHP checks:

$$(\mathsf{Id}_{k_1} \otimes \Gamma^{(2)}) \odot \boldsymbol{\gamma^{(1)}} \stackrel{?}{=} (\Gamma^{(1)} \otimes \mathsf{Id}_{k_2}) \odot \boldsymbol{\gamma^{(2)}}, \tag{3}$$

where $\boldsymbol{\gamma^{(1)}} = (\gamma_j)_{j=1,\ldots,n_2 k_1}$ and $\boldsymbol{\gamma^{(2)}} = (\gamma_j)_{j=n_2 k_1+1, n_2 k_1 + n_1 k_2}$.

Let $C_1 \in \mathscr{L}_1$, $C_2 \in \mathscr{L}_2$ with $\boldsymbol{\lambda_1}$ and $\boldsymbol{\lambda_2}$ such that $\hat{\boldsymbol{C}}_1 = \boldsymbol{\lambda_1} \odot \Gamma^{(1)}$ and $\hat{\boldsymbol{C}}_2 = \boldsymbol{\lambda_2} \odot \Gamma^{(2)}$. The hash value computed with ProjHash is then:

$$H' = (\boldsymbol{\lambda_1} \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\gamma^{(1)}} = ((\boldsymbol{\lambda_1} \odot \mathsf{Id}_{k_1}) \otimes (\boldsymbol{\lambda_2} \odot \Gamma^{(2)})) \odot \boldsymbol{\gamma^{(1)}} = (\boldsymbol{\lambda_1} \otimes \boldsymbol{\lambda_2}) \odot (\mathsf{Id}_{k_1} \otimes \Gamma^{(2)}) \odot \boldsymbol{\gamma^{(1)}}$$

while the hash value computed with THash is:

$$H'' = (\hat{\boldsymbol{C}}_1 \otimes \boldsymbol{\lambda_2}) \odot \boldsymbol{\gamma^{(2)}} = ((\boldsymbol{\lambda_1} \odot \Gamma^{(1)}) \otimes (\boldsymbol{\lambda_2} \odot \mathsf{Id}_{k_2})) \odot \boldsymbol{\gamma^{(2)}} = (\boldsymbol{\lambda_1} \otimes \boldsymbol{\lambda_2}) \odot (\Gamma^{(1)} \otimes \mathsf{Id}_{k_2}) \odot \boldsymbol{\gamma^{(2)}}.$$

And so $H' = H''$ and Equation (3) if verified, when hp is valid.

Comparison with the Original Construction. If $\mathscr{L}_2$ is just the language of DDH tuples (as in Example 1), then we get a TSPHF slightly less efficient than the original construction: the second part of the projection key ($\boldsymbol{\gamma^{(2)}}$ here and χ in [BBC+13]) is twice as long. However, the advantage is that our construction works in more cases: it does not require that there is a way to generate a CRS crs with a trapdoor $\mathcal{T}_{\mathsf{crs}}$ enabling to compute the discrete logarithms of elements of Γ_1, but only enabling to check if a word C_1 is in $\mathscr{L}_1$ or not.

We will also see in Section 9, that the original construction can also be seen as a disjunction of two diverse vector spaces, but the second diverse vector space will be one for which $\mathscr{L}_2 = \mathcal{X}_2$.

7 Pseudo-Random Projective Hash Functions and Disjunctions

7.1 Pseudo-Randomness

Definition. An SPHF is said to be *pseudo-random*, if the hash value of a random word C in $\mathscr{L}_{\mathsf{crs},\mathsf{par}}$ looks random to an adversary only knowing the projection key hp and ignoring the hashing key hk and a witness for the word C. More precisely, this property is defined by the experiments $\mathsf{Exp}^{\texttt{ps-rnd-}b}$ depicted in Fig. 3. Contrary to smoothness, this property is computational.

A projective hashing function which is pseudo-random is called a PrPHF. A PrPHF is not necessarily smooth.

$\mathsf{Exp}^{\texttt{ps-rnd-}b}(\mathcal{A}, \mathfrak{K})$
  $(\mathsf{crs}, \mathcal{T}_{\mathsf{crs}}) \xleftarrow{\$} \mathsf{Setup}_{\mathsf{crs}}(1^{\mathfrak{K}})$
  $\mathsf{hk} \xleftarrow{\$} \mathsf{HashKG}(\mathsf{crs})$
  $\mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{crs})$
  $(\mathsf{par}, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(\mathsf{crs})$
  $C \xleftarrow{\$} \mathscr{L}_{\mathsf{crs},\mathsf{par}}$
  if $b = 0$ **then**
    $H \leftarrow \mathsf{Hash}(\mathsf{hk}, (\mathsf{crs}, \mathsf{par}), C)$
  else
    $H \xleftarrow{\$} \Pi$
  return $\mathcal{A}(\mathsf{st}, H)$

Fig. 3. Experiments $\mathsf{Exp}^{\texttt{ps-rnd-}b}$ for pseudo-randomness

Link with Hard Subset Membership Languages. It is easy to see that an SPHF over a hard subset membership family of languages is pseudo-random.

This yields a way to create PrPHF under DDH using Example 1. However, this is inefficient since, in this case $\mathcal{X}$ has dimension 2, while we would prefer to have $\mathcal{X}$ of dimension 1. Actually, since for hard subset membership languages, $\mathscr{L}_{\mathsf{crs},\mathsf{par}} \neq \mathcal{X}$, any SPHF based on diverse vector space for such languages is such that $\mathcal{X}$ has dimension at least 2. More generally, as shown in 5.5, for a hard subset membership language based on κ-Lin, $\mathcal{X} = \mathbb{G}^{1\times(\kappa+1)}$ and $\mathscr{L}_{\mathsf{crs}}$ has dimension κ. That is why, we introduce another way to construct PrPHF, still based on diverse vector spaces, but not using hard subset membership languages.

7.2 Canonical PrPHF under κ-Lin

Let us construct a diverse vector space $(\mathcal{X}, (\mathscr{L}_{\mathsf{crs}}), \mathbb{G}, n, k, \Gamma, (\theta_{\mathsf{crs}}))$ which yields a pseudo-random SPHF under κ-Lin in the cyclic group $\mathbb{G}$. par is not used in this section and is therefore dropped. In the following sections, $\mathbb{G}$ will often be seen as a sub-graded ring of some graded ring $\mathfrak{G}$.

We set $\mathcal{X} = \mathscr{L}_{\mathsf{crs}} = \{\perp\}$ and $\hat{\mathcal{X}} = \hat{\mathscr{L}}_{\mathsf{crs}} = \mathbb{G}^{1\times k}$. For DDH = 1-Lin, we gen a PrPHF with $\mathcal{X}$ of dimension 1, which is the best we can do using diverse vector spaces. Even though the resulting projective hash function will be smooth, the smoothness property is completely trivial, since $\mathscr{L}_{\mathsf{crs}} \setminus \mathcal{X}$ is empty, and does not imply the pseudo-randomness property. We will therefore need to manually prove the pseudo-randomness.

Construction. The "language" is defined by $\mathsf{crs} = (\zeta_1, \ldots, \zeta_\kappa) \xleftarrow{\$} \mathbb{G}^\kappa$. Γ and θ are defined as follows:

$$\Gamma = \begin{pmatrix} \zeta_1 & 0 & \ldots & 0 \\ 0 & \zeta_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \zeta_\kappa \end{pmatrix} \in \mathbb{G}^{\kappa\times\kappa} \qquad\qquad \theta(\perp) = (g, \ldots, g) \in \mathbb{G}^{1\times\kappa}$$

so that the witness $\boldsymbol{\lambda}$ for $C = \perp$ is the vector $\boldsymbol{\lambda} = (\hat{\zeta}_1, \ldots, \hat{\zeta}_\kappa) \in \mathbb{Z}_p^{1\times\kappa}$, where $\zeta_i = g^{1/\hat{\zeta}_i}$.

Concretely, this gives:

$$\mathsf{hk} := \boldsymbol{\alpha} \xleftarrow{\$} \mathbb{Z}_p^\kappa \quad \mathsf{hp} := (\gamma_1, \ldots, \gamma_\kappa) = (\zeta_1^{\alpha_1}, \ldots, \zeta_\kappa^{\alpha_\kappa}) \quad H := \prod_{i=1}^n g^{\alpha_i} = g^{\sum_{i=1}^n \alpha_i} = \prod_{i=1}^n \gamma_i^{\hat{\zeta}_i} =: H'.$$

Proof of Pseudo-Randomness. The pseudo-randomness directly comes from the hardness of κ-Lin.

7.3 Disjunction of an SPHF and a PrPHF

Let $\mathcal{V}_1 = (\mathcal{X}_1, (\mathscr{L}_{1,\mathsf{crs}_1,\mathsf{par}}), \mathfrak{G}_1, n_1, k_1, \Gamma^{(1,\mathsf{crs}_1)}, (\theta_{1,\mathsf{crs}_1,\mathsf{par}}))$ and $\mathcal{V}_2 = (\mathcal{X}_2, (\mathscr{L}_{2,\mathsf{crs}_2}), \mathfrak{G}_2, n_2, k_2, \Gamma^{(2,\mathsf{crs}_2)}, (\theta_{2,\mathsf{crs}_2}))$ be two diverse vector spaces over two multiplicatively sub-graded rings $\mathfrak{G}_1$ and $\mathfrak{G}_2$ of some graded ring $\mathfrak{G}$. Let $(\mathcal{X}, (\mathscr{L}_{\mathsf{crs},\mathsf{par}}), \mathfrak{G}, n, k, \Gamma^{(\mathsf{crs})}, (\theta_{\mathsf{crs},\mathsf{par}}))$ be the vector space corresponding to the disjunction of the two previous languages. We have already seen that this vector space corresponds to a smooth projective hash function.

But, if the second language is the canonical PrPHF under κ-Lin, the smoothness brings nothing, since $\mathcal{X} = \mathscr{L}$. Therefore, we need to prove a stronger property called *mixed pseudo-randomness.*

Definition of Mixed Pseudo-Randomness. The resulting SPHF is said mixed pseudo-random, if the hash value of a word $C = (C_1, C_2)$ looks random to the adversary, when par and $C_1 \notin \mathscr{L}_{1,\mathsf{crs}_1,\mathsf{par}}$ are chosen by the adversary, while C_2 is chosen at random in $\mathscr{L}_{2,\mathsf{crs}_2}$. More precisely, the mixed pseudo-randomness property is defined by the experiments $\mathsf{Exp}^{\texttt{mixed-ps-rnd-}b}$ depicted in Fig. 4.

$\mathsf{Exp}^{\texttt{mixed-ps-rnd-}b}(\mathcal{A}, \mathfrak{K})$
- $(\mathsf{crs} = (\mathsf{crs}_1, \mathsf{crs}_2), (\mathcal{T}_{\mathsf{crs}_1}, \mathcal{T}_{\mathsf{crs}_2})) \xleftarrow{\$} \mathsf{Setup}_{\mathsf{crs}}(1^{\mathfrak{K}})$
- $\mathsf{hk} \xleftarrow{\$} \mathsf{HashKG}(\mathsf{crs})$
- $\mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{crs})$
- $C_2 \xleftarrow{\$} \mathscr{L}_{2,\mathsf{crs}_2}$
- $(\mathsf{par}, C_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}, \mathcal{T}_{\mathsf{crs}_1}, C_2)$
- $C \leftarrow (C_1, C_2)$
- **if** $b = 0$ or $C_1 \in \mathscr{L}_{1,\mathsf{crs}_1,\mathsf{par}}$ **then**
  - $H \leftarrow \mathsf{Hash}(\mathsf{hk}, (\mathsf{crs}, \mathsf{par}), C)$
- **else**
  - $H \xleftarrow{\$} \Pi$
- **return** $\mathcal{A}(\mathsf{st}, H)$

Fig. 4. Experiment $\mathsf{Exp}^{\texttt{mixed-ps-rnd-}b}$ for mixed pseudo-randomness

Proof of Mixed Pseudo-Randomness. The proof of mixed pseudo-randomness is actually close to the one for computational soundness of trapdoor smooth projective functions in [BP13]. It requires that $\mathcal{T}_{\mathsf{crs}_1}$ contains enough information to be able to compute the discrete logarithm of elements of $\Gamma^{(1)}$, denoted $\mathfrak{L}(\Gamma^{(1)})$.

The proof reduces the pseudo-randomness property to the mixed pseudo-randomness property. The detailed proof is quite technical and can be found in Appendix B. Basically, we choose a random hashing key $\boldsymbol{\varepsilon}$ and we randomize it using a basis of the kernel of $\mathfrak{L}(\Gamma^{(1)})$ and projection keys given by the pseudo-randomness game (for some fixed word C_2, using an hybrid method). Then we show how to compute from that, a valid projection key hp for the language of the disjunction together with a hash value H of (C_1, C_2), for $C_1 \notin \mathscr{L}_1$. This value H is the correct hash value, if the hash values of C_2, given by the challenger of the hybrid pseudo-randomness game, were valid; and it is a random value, otherwise. That proves that an adversary able to break the mixed pseudo-randomness property also breaks the pseudo-randomness property.

8 One-Time Simulation-Sound NIZK from Disjunctions of an SPHF and a PrPHF

8.1 NIZK from Disjunctions of a SPHF and an PrPHF

The construction is identical to the one in Section 5.2, except that the second diverse vector space $\mathcal{V}_2$ is just supposed to be a PrPHF, and no more supposed to be related to a hard subset membership language $\mathscr{L}_2$. However, we suppose that the disjunction of $\mathcal{V}_1$ and $\mathcal{V}_2$ yield a mixed pseudo-random SPHF, which is the case if $\mathcal{T}_{\mathsf{crs}}$ contains enough information to compute the discrete logarithm of elements of $\Gamma^{(1)}$.

Completeness and zero-knowledge can be proven exactly in the same way. It remains therefore to prove the soundness property, under the mixed pseudo-randomness. The proof is very similar to the one in Section 5.2: if $\boldsymbol{\pi}$ is a proof of some word $C_1 \notin \mathscr{L}_1$, then it is possible to compute the hash value of any word (C_1, C_2) with $C_2 \in \mathscr{L}_2$ as:

$$H' := \hat{\boldsymbol{C}}_2 \odot \boldsymbol{\pi}.$$

This comes from the fact that if $C_2 \in \mathscr{L}_2$, then there exists $\boldsymbol{\lambda_2}$ such that $\hat{\boldsymbol{C}}_2 = \boldsymbol{\lambda_2} \odot \Gamma^{(2)}$, hence:

$$H' = \boldsymbol{\lambda_2} \odot \Gamma^{(2)} \odot \boldsymbol{\pi} = \boldsymbol{\lambda_2} \odot (\hat{\boldsymbol{C}}_1 \otimes \mathsf{Id}_{k_2}) \odot \boldsymbol{\gamma^{(2)}} = (\hat{\boldsymbol{C}}_1 \otimes \boldsymbol{\lambda_2}) \odot \boldsymbol{\gamma^{(2)}},$$

which is the hash value of (C_1, C_2) computed using $\mathsf{ProjHash}$ and witness $\boldsymbol{\lambda_2}$. But the mixed pseudo-randomness property ensures that this values looks uniformly random when C_2 is chosen randomly in $\mathscr{L}_2$. That proves the soundness property.

8.2 One-Time Simulation-Sound NIZK

Unfortunately, for the one-time simulation-sound variant, this is not as easy: the construction in Section 5.4 seems difficult (if at all possible) to prove sound. The main problem is that the security proof of mixed pseudo-randomness is not statistical, so we do not know $\mathsf{hk} = \boldsymbol{\alpha}$, but only some representation of $\boldsymbol{\alpha}$, which does not allow computing the proof $\boldsymbol{\pi'}$ of a word C_1' for a tag $\mathsf{tag}_{C_1'}$. Directly adapting the proof with a 2-smooth $\mathcal{V}_1$ would require to choose from the beginning $\boldsymbol{\pi'}$ (as is chosen hp from the beginning), but that is not possible since C_1' and tag' (the tag for C_1') are not known at the beginning of the game.

So the idea is to use the tag bit-by-bit. So we just need to guess which bit is different between tag_{C_1} and $\mathsf{tag}_{C_1'}$. This idea is inspired from [CW13]. More precisely, we show how to transform $\mathcal{V}_1$ into another diverse vector space $\tilde{\mathcal{V}}_1$ such that the disjunction of $\tilde{\mathcal{V}}_1$ and $\mathcal{V}_2$ yields a one-time simulation-sound NIZK.

Let us suppose tags are binary strings of length ν: $\mathsf{Tags} = \{0,1\}^\nu$. tag_i represents the bit $i \in \{1, \dots, \nu\}$ of $\mathsf{tag} \in \mathsf{Tags}$. We transform the original diverse vector space $\mathcal{V}_1$ for $\mathscr{L}_1$ (not the 2-smooth one) into $\tilde{\mathcal{V}}_1 = (\mathcal{X}, (\mathscr{L}_{\mathsf{par}}), \mathfrak{G}_1, \tilde{n_1}, \tilde{k_1}, \tilde{\Gamma^{(1)}}, (\tilde{\theta}_{1,\mathsf{par}}))$ with:

$$\begin{aligned}
\tilde{n_1} &= 2\nu n_1 \\
\tilde{k_1} &= 2\nu k_1 \\
\tilde{\Gamma} &= \begin{pmatrix} \Gamma & 0 & \dots & 0 \\ 0 & \Gamma & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \Gamma \end{pmatrix} \\
\tilde{\theta}_{\mathsf{par}}(C, \mathsf{tag}) &= ((1 - \mathsf{tag}_1) \odot \hat{\boldsymbol{C}},\, \mathsf{tag}_1 \odot \hat{\boldsymbol{C}},\, \dots,\, (1 - \mathsf{tag}_\nu) \odot \hat{\boldsymbol{C}}, \mathsf{tag}_\nu \odot \hat{\boldsymbol{C}}).
\end{aligned}$$

Now, we can construct a one-time simulation NIZK exactly as the NIZK from disjunction of an SPHF and a PrPHF in Section 8.1, except that $\mathcal{V}_1$ is replaced by $\tilde{\mathcal{V}}_1$. Completeness and perfect unbounded zero-knowledge are straightforward. Let us now prove that the one-time simulation soundness property.

We suppose that the adversary asks for a simulated proof $\boldsymbol{\pi'}$ for some word C_1' and some label ℓ', and we write $\mathsf{tag}' = \mathcal{H}((C_1', \ell'))$; then the adversary submits a proof $\boldsymbol{\pi}$ for a word $C_1' \notin \mathscr{L}_1$ and some label ℓ, and we write $\mathsf{tag} = \mathcal{H}((C_1, \ell))$. Thanks to the collision resistance of $\mathcal{H}$, we can assume that $\mathsf{tag}' \neq \mathsf{tag}$.

We then remark that we can write any hashing key $\tilde{\mathsf{hk}}$ for the disjunction of $\tilde{\mathcal{V}}_1$ and $\mathcal{V}_2$ as the concatenation of 2ν hashing keys for the disjunction of $\mathcal{V}_1$ and $\mathcal{V}_2$: $\hat{\mathsf{hk}}_1, \dots, \hat{\mathsf{hk}}_{2\nu}$ and that the hash value of C for tag and $\tilde{\mathsf{hk}}$ is just the product of the hash values of C for all the $\hat{\mathsf{hk}}_{2i+\mathsf{tag}_i-1}$.

Therefore, we guess an index i and a bit b such that $\mathsf{tag}_i = b$ but $\mathsf{tag}'_i = 1-b$. If this guess is wrong, we just abort the reduction. Our guess will be correct with probability at least $1/(2\nu)$ which makes our reduction polynomial time.

Finally, we just need to remark that the mixed pseudo-randomness ensures that the hash value H of C for tag under $\hat{\mathsf{hk}}_{2i+\mathsf{tag}_i-1}$ looks random, since $\hat{\mathsf{hk}}_{2i+\mathsf{tag}_i-1}$ is only used to compute H and nothing else: the simulated proof $\boldsymbol{\pi'}$ is the hash value of $\hat{\boldsymbol{C}}'_1 \otimes \mathsf{Id}_{n_2}$ with $\mathsf{tag}'_i \neq b$, so $\hat{\mathsf{hk}}_{2i+\mathsf{tag}_i-1}$ is not used to compute it. This shows that H is uniformly random.

8.3 Concrete Instantiation and Comparison with Existing Works

If $\mathcal{V}_1$ is a diverse vector space over $\mathbb{G}_1$ (for which $\mathcal{T}_{\mathsf{crs}_1}$ gives enough information to compute the discrete logarithm of $\Gamma^{(1)}$) and $\mathcal{V}_2$ is the canonical PrPHF under DDH in Section 7.2, where $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ is a bilinear group where DDH is hard in $\mathbb{G}_2$, then we get an NIZK and a one-time simulation sound NIZK whose proof is composed of only $n_2 = 1$ group element in $\mathbb{G}_1$.

More generally, if $\mathcal{V}_2$ is canonical PrPHF under κ-Lin, then the proof consists of only κ group elements, one less than our first construction in Section 5.5. However, this encompasses slightly fewer languages than this first construction, due to the restriction on $\mathscr{L}_1$ and $\mathcal{T}_{\mathsf{crs}_1}$. More precisely, our NIZK handles the same languages as Jutla-Roy NIZK in [JR13, JR14].

Table 1 compares NIZK for linear subspaces as Jutla and Roy call it in [JR13], i.e., any language over $\mathbb{G}_1$ (first group of some bilinear group) for which there exists a diverse vector space $\mathcal{V}_1$ (assuming θ is the identity function and a witness is $\boldsymbol{\lambda} \in \mathbb{Z}_p^k$). Some of the entries of this table were derived from [JR14] and from [LPJY14]. The DDH (in $\mathbb{G}_2$) variant requires asymmetric bilinear groups, while the κ-Lin variant for $\kappa \geq 2$ could work on symmetric bilinear groups. DLin corresponds to 2-Lin.

First of all, as far as we know, our one-time simulation-sound NIZK is the most efficient such NIZK with a constant-size proof: the single-theorem relatively sound construction of Libert et al. [LPJY14] is weaker than our one-time simulation-sound NIZK and requires at least one group element more in the proof, while their universal simulation-sound construction is much more inefficient. In addition, their constructions only work in symmetric bilinear groups, while ours work both in asymmetric and symmetric bilinear groups. A direct application of our construction is our efficient structure-preserving threshold $\mathsf{IND\text{-}CCA}$ encryption scheme, under DDH.

Second, the DLin version of our NIZK in Section 5.2 is similar to the one by Libert et al. [LPJY14], but our DLin version of our NIZK in Section 8.1 is more efficient (the proof has 2 group elements instead of 3). Furthermore, the ideas of the constructions in [LPJY14] seem quite different, and our ideas carry on asymmetric bilinear group (under DDH).

Third, our NIZK in Section 8.1 is similar to the one by Jutla and Roy in [JR14] for DDH. However, in our opinion, our construction seems to be more modular and simpler to understand. In addition, under κ-Lin, with $\kappa \geq 2$, our construction is slightly more efficient in terms of CRS size and verification time.

8.4 Application: Threshold Cramer-Shoup-like Encryption Scheme (Variant)

In the construction of Section 5.6, we can replace the previous one-time simulation-sound NIZK by this new NIZK. This yields a threshold encryption where the ciphertext size only consists of 4 group elements as the original Cramer-Shoup encryption scheme, at the expense of having a public key size linear in the security parameter.

A comparison with existing efficient $\mathsf{IND\text{-}CCA}$ encryption schemes based on cyclic or bilinear groups is given in Table 2, whose entries have been partially derived from similar tables in [BMW05, Kil06].

Table 1. Comparison of NIZK for linear subspaces

		WS	DDH (in $\mathbb{G}_2$) Proof $\lvert\pi\rvert$	CRS $\lvert\sigma\rvert$	Pairings	DLin (in $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$) Proof $\lvert\pi\rvert$	CRS $\lvert\sigma\rvert$	Pairings
[GS08]			$n + 2k$	5	$2n(k + 2)$	$2n + 3k$	6	$3n(k + 3)$
[JR13]		yes	$n - k$	$2k(n - k) + 2$	$(n - k)(k + 2)$	$2n - 2k$	$4k(n - k) + 3$	$2(n - k)(k + 2)$
[LPJY14]						3	$2n + 3k + 3$	$2n + 4$
[LPJY14]	RSS					4	$4n + 8t + 5$	$2n + 6$
[LPJY14]	USS					20	$2n + 3t + 3\nu + 10$	$2n + 30$
[JR14]		yes	1	$n + k + 1$	$n + 1$	2	$2(n + k + 2)$	$2(n + 2)$
Ours §5.2			2	$n + 2k + 1$	$n + 2$	3	$2n + 3k + 2$	$2n + 3$
Ours §8.1		yes	1	$n + k + 1$	$n + 1$	2	$2n + 2k + 2$	$2n + 2$
Ours §5.4	OTSS		2	$2(n + 2k) + 1$	$2n + 2$	3	$2(2n + 3k) + 2$	$4n + 3$
Ours §8.2	OTSS	yes	1	$2\nu(2n + 3k) + 2$	$\nu n + 2$	2	$2\nu(2n + 3k) + 2$	$2\nu n + 2$

- $n = n_1$, $k = k_1$, and $\nu = 2\mathfrak{K}$;
- sizes $\lvert \cdot \rvert$ are measured in term of group elements ($\mathbb{G}_1$ and $\mathbb{G}_2$, or $\mathbb{G}$ if the bilinear group is symmetric). Generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ (for DDH in $\mathbb{G}_2$) or $g \in \mathbb{G}$ (for DLin) are not counted in the CRS;
- OTSS: one-time simulation-soundness;
- RSS: single-theorem relative simulation-soundness [JR12] (weaker than OTSS);
- USS: universal simulation-soundness (stronger than OTSS);
- WS: witness-samplability in [JR13], generation of crs so that $\mathcal{T}_{\mathsf{crs}_1}$ enables to compute the discrete logarithms of Γ. This slightly restrict the set of languages which can be handled;
- Proof and CRS are the sizes $\lvert\pi\rvert$ and $\lvert\sigma\rvert$, in term of group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ (generators g_1 and g_2 are not counted)
- Pairings: number of pairings required to verify the proof.

Our two schemes are threshold and *structure-preserving* [AFG+10]: they are "compatible" with Groth-Sahai NIZK, in the sense that we can do a Groth-Sahai NIZK to prove that we know the plaintext of a ciphertext for our encryption schemes. In addition, normal decryption does not require any pairings, which still are very costly, compared to exponentiations.

The two other efficient threshold and structure-preserving IND-CCA encryption schemes are those based on the Canetti-Halevi-Katz [CHK04] transform, the one of Boyen, Mei and Waters [BMW05] and the one of Kiltz [Kil06]. But for all except the one of Kiltz, the plaintext and one element of the ciphertext has to be in $\mathbb{G}_T$, which limits usage of Groth-Sahai NIZK. In addition, elements in $\mathbb{G}_T$ have a much longer representation than elements in $\mathbb{G}$, $\mathbb{G}_1$ or $\mathbb{G}_2$. And, even though our second encryption scheme uses exactly the same number of group elements as Kiltz's encryption scheme [Kil06], these groups elements are about 50% smaller in practice, since we use an asymmetric pairing while Kiltz's scheme uses a symmetric one. So even our first construction is more efficient (regarding ciphertext size) than Kiltz's construction.

To summarize, to the best of our knowledge, our two constructions are the most efficient threshold and structure-preserving IND-CCA encryption schemes.

9 Building TSPHFs from Disjunctions of SPHFs and PrPHFs

It is interesting to notice that one can obtain the original TSPHF in [BBC+13] by simply replacing the second hard membership diverse vector space $\mathcal{V}_2$ by the canonical PrPHF $\mathcal{V}_2$ under κ-Lin in $\mathbb{G}_2$ in the construction in Section 6.2. Even though this observation does not lead to a performance improvement, it sheds a new light into the way the original TSPHF construction works, namely, that it was just a disjunction of the language $\mathscr{L}_1$ and a trivial language $\mathscr{L}_2$.

As previously stated, the original SPHF in [BBC+13] is more efficient than our new SPHF construction in Section 6.2. However, while the original construction requires the generation of a trapdoor $\mathcal{T}_{\mathsf{crs}}$ for computing the discrete logarithm of Γ, our new SPHF is less restrictive since it only requires a trapdoor $\mathcal{T}_{\mathsf{crs}}$ for checking whether a word C_1 is in $\mathscr{L}_1$ or not.

Table 2. Efficiency comparison for IND-CCA encryption schemes over cyclic or bilinear groups

		Time Complexity[a]			Ciphertext Overhead		
Scheme	Assumption	Encryption	Decryption	Public key	Hybrid	SP[b]	Thres.[c]
KD	DDH	0 + [1,2,0]	0 + [1,0,0]	4 $\mathbb{G}$	2 $\mathbb{G}$ (+hybrid)	n/a	✓
CS	DDH	0 + [1,3,0]	0 + [1,1,0]	5 $\mathbb{G}$	3 $\mathbb{G}$	+ $\mathbb{G}$	
CHK/BB1	BDDH	0 + [1,2,0]	1 + [1,0,0]	$O(1)$[d]	2 $\mathbb{G}$ + ver key + sig	+ $\mathbb{G}_T$	✓
CHK/BB2	q-BDDHI	0 + [1,2,0]	1 + [0,1,1]	$O(1)$[d]	2 $\mathbb{G}$ + ver key + sig	+ $\mathbb{G}_T$	✓
BK/BB1	BDDH	0 + [1,2,0]	1 + [1,0,0]	$O(1)$[d]	2 $\mathbb{G}$ + com + mac	+ $\mathbb{G}_T$	
BK/BB2	q-BDDHI	0 + [1,2,0]	1 + [0,1,1]	$O(1)$[d]	2 $\mathbb{G}$ + com + mac	+ $\mathbb{G}_T$	
BMW	BDDH	0 + [1,2,0]	1 + [0,1,0]	2 $\mathbb{G}$ + $\mathbb{G}_T$	2 $\mathbb{G}$	+ $\mathbb{G}_T$	✓
Kiltz	DLin	0 + [2,3,0]	0 + [1,0,0]	5 $\mathbb{G}$	4 $\mathbb{G}$[e]	+ $\mathbb{G}$[e]	✓
Ours §5.6[f]	SXDH	0 + [3,1,0]	0 + [2,1,0]	6 $\mathbb{G}_1$	4 $\mathbb{G}_1$	+ $\mathbb{G}_1$	✓
Ours §8.4	SXDH	0 + [1,2,0] + $2\mathfrak{K}$	0 + [1,1,0] + $2\mathfrak{K}$	$(3 + 4\mathfrak{K})$ $\mathbb{G}_1$	3 $\mathbb{G}_1$	+ $\mathbb{G}_1$	✓

KD: Kurosawa-Desmedt [KD04], CS: Cramer-Shoup [CS98], CHK: Canetti-Halevi-Katz transform [CHK04] for BB1/BB2 Boneh-Boyen IBE [BB04], BK: Boneh-Katz transformation [BK05], BMW: Boneh-Mey-Waters [BMW05], Kiltz [Kil06]

ver key: verification key of a signature scheme, sig: signature, com: commitment

[a] #pairing + [#multi, #regular, #fix]-exponentiation (+ #multiplication) (in $\mathbb{G}$ or $\mathbb{G}_1$), a multi-exponentiation being an exponentiation of two group elements in the same time; the number of multiplications is approximate and only written when it depends on $\mathfrak{K}$, since multiplications are way faster than pairings and exponentiations;

[b] Number of other elements required to make the KEM (previous column) scheme, a structure preserving encryption scheme; see text;

[c] support threshold decryption;

[d] depends on parameters for the signature/commitment/mac and if we use symmetric or asymmetric groups, but a small constant in any case;

[e] $\mathbb{G}$ has to be a cyclic group from a symmetric bilinear group, and so element representation is often 50% bigger than for the other scheme where $\mathbb{G}$ is either just a cyclic group, or can be the first group ($\mathbb{G}_1$) of an asymmetric bilinear group;

[f] supposing $\nu = 2\mathfrak{K}$.

Acknowledgments

This work was supported in part by the French ANR-12-INSE-0014 SIMPATIC Project, the CFM Foundation, the European Commission through the FP7-ICT-2011-EU-Brazil Program under Contract 288349 SecFuNet, and the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud).

References

ABB+13. M. Abdalla, F. Benhamouda, O. Blazy, C. Chevalier, and D. Pointcheval. SPHF-friendly non-interactive commitments. In *ASIACRYPT 2013, Part I*, *LNCS* 8269, pages 214–234. Springer, December 2013. (Page 1.)

ABCP06. M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval. Password-based group key exchange in a constant number of rounds. In *PKC 2006*, *LNCS* 3958, pages 427–442. Springer, April 2006. (Pages 2, 23, 41, and 42.)

ACP09. M. Abdalla, C. Chevalier, and D. Pointcheval. Smooth projective hashing for conditionally extractable commitments. In *CRYPTO 2009*, *LNCS* 5677, pages 671–689. Springer, August 2009. (Pages 1, 3, 5, and 7.)

AFG+10. M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In *CRYPTO 2010*, *LNCS* 6223, pages 209–236. Springer, August 2010. (Page 31.)

AFP05. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In *PKC 2005*, *LNCS* 3386, pages 65–84. Springer, January 2005. (Page 41.)

BB04. D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *EUROCRYPT 2004*, *LNCS* 3027, pages 223–238. Springer, May 2004. (Page 32.)

BBC+13. F. Benhamouda, O. Blazy, C. Chevalier, D. Pointcheval, and D. Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. In *CRYPTO 2013, Part I*, *LNCS* 8042, pages 449–475. Springer, August 2013. (Pages 1, 2, 3, 4, 5, 6, 7, 9, 12, 13, 15, 22, 24, 26, 31, and 42.)

BK05. D. Boneh and J. Katz. Improved efficiency for CCA-secure cryptosystems built using identity-based encryption. In *CT-RSA 2005*, *LNCS* 3376, pages 87–103. Springer, February 2005. (Page 32.)

BMW05. X. Boyen, Q. Mei, and B. Waters. Direct chosen ciphertext security from identity-based techniques. In *ACM CCS 05*, pages 320–329. ACM Press, November 2005. (Pages 30, 31, and 32.)

BP13. F. Benhamouda and D. Pointcheval. Trapdoor smooth projective hash functions. Cryptology ePrint Archive, Report 2013/341, 2013. `http://eprint.iacr.org/2013/341`. (Page 28.)

BPR00. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, *LNCS* 1807, pages 139–155. Springer, May 2000. (Page 2.)

BPV12. O. Blazy, D. Pointcheval, and D. Vergnaud. Round-optimal privacy-preserving protocols with smooth projective hash functions. In *TCC 2012*, *LNCS* 7194, pages 94–111. Springer, March 2012. (Page 1.)

BS03. D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003. (Page 24.)

CHK04. R. Canetti, S. Halevi, and J. Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT 2004*, *LNCS* 3027, pages 207–222. Springer, May 2004. (Pages 31 and 32.)

CLT13. J.-S. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In *CRYPTO 2013, Part I*, *LNCS* 8042, pages 476–493. Springer, August 2013. (Pages 4 and 13.)

CS98. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO'98*, *LNCS* 1462, pages 13–25. Springer, August 1998. (Pages 1, 4, 22, 32, 36, and 40.)

CS02. R. Cramer and V. 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. (Pages 1, 3, 6, 12, 15, and 20.)

CW13. J. Chen and H. Wee. Fully, (almost) tightly secure IBE and dual system groups. In *CRYPTO 2013, Part II*, *LNCS* 8043, pages 435–460. Springer, August 2013. (Page 29.)

ElG84. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO'84*, *LNCS* 196, pages 10–18. Springer, August 1984. (Page 35.)

GGH13. S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT 2013*, *LNCS* 7881, pages 1–17. Springer, May 2013. (Pages 4 and 13.)

GGSW13. S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. In *45th ACM STOC*, pages 467–476. ACM Press, June 2013. (Pages 2 and 3.)

GL03. R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. In *EUROCRYPT 2003*, *LNCS* 2656, pages 524–543. Springer, May 2003. `http://eprint.iacr.org/2003/032.ps.gz`. (Pages 1 and 2.)

Gro06. J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT 2006*, *LNCS* 4284, pages 444–459. Springer, December 2006. (Pages 3 and 17.)

GS08. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT 2008*, *LNCS* 4965, pages 415–432. Springer, April 2008. (Pages 4 and 31.)

JR12. C. S. Jutla and A. Roy. Relatively-sound NIZKs and password-based key-exchange. In *PKC 2012*, *LNCS* 7293, pages 485–503. Springer, May 2012. (Pages 1 and 31.)

JR13. C. S. Jutla and A. Roy. Shorter quasi-adaptive NIZK proofs for linear subspaces. In *ASIACRYPT 2013, Part I*, *LNCS* 8269, pages 1–20. Springer, December 2013. (Pages 4, 17, 22, 30, and 31.)

JR14. C. S. Jutla and A. Roy. Switching lemma for bilinear tests and constant-size nizk proofs for linear subspaces, August 2014. (Pages 5, 30, and 31.)

Kal05. Y. T. Kalai. Smooth projective hashing and two-message oblivious transfer. In *EUROCRYPT 2005*, *LNCS* 3494, pages 78–95. Springer, May 2005. (Page 1.)

KD04. K. Kurosawa and Y. Desmedt. A new paradigm of hybrid encryption scheme. In *CRYPTO 2004*, *LNCS* 3152, pages 426–442. Springer, August 2004. (Page 32.)

Kil06. E. Kiltz. Chosen-ciphertext security from tag-based encryption. In *TCC 2006*, *LNCS* 3876, pages 581–600. Springer, March 2006. (Pages 30, 31, and 32.)

KOY01. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001*, *LNCS* 2045, pages 475–494. Springer, May 2001. (Pages 1 and 2.)

KV11. J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. In *TCC 2011*, *LNCS* 6597, pages 293–310. Springer, March 2011. (Pages 1, 2, 9, 24, 42, and 43.)

LPJY14. B. Libert, T. Peters, M. Joye, and M. Yung. Non-malleability from malleability: Simulation-sound quasi-adaptive NIZK proofs and CCA2-secure encryption from homomorphic signatures. In *EUROCRYPT 2014*, *LNCS* 8441, pages 514–532. Springer, May 2014. (Pages 5, 30, and 31.)

SG02. V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, 2002. (Page 40.)

Sha79. A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. (Page 40.)

Sha07. H. Shacham. A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants. Cryptology ePrint Archive, Report 2007/074, 2007. `http://eprint.iacr.org/`. (Pages 24 and 36.)

A Notations

A.1 Distances, Advantage and Success

Statistical Distance. Let $\mathcal{D}_0$ and $\mathcal{D}_1$ be two probability distributions over a finite set $\mathcal{S}$ and let X_0 and X_1 be two random variables with these two respective distributions. The statistical distance between $\mathcal{D}_0$ and $\mathcal{D}_1$ is also the statistical distance between X_0 and X_1:

$$\mathtt{Dist}(\mathcal{D}_0, \mathcal{D}_1) = \mathtt{Dist}(X_0, X_1) = \sum_{x \in \mathcal{S}} |\Pr[X_0 = x] - \Pr[X_1 = x]|.$$

If the statistical distance between $\mathcal{D}_0$ and $\mathcal{D}_1$ is less than or equal to ε, we say that $\mathcal{D}_0$ and $\mathcal{D}_1$ are ε-close or are ε-statistically indistinguishable. If the $\mathcal{D}_0$ and $\mathcal{D}_1$ are 0-close, we say that $\mathcal{D}_0$ and $\mathcal{D}_1$ are perfectly indistinguishable.

Success/Advantage. When one considers an experiment $\mathsf{Exp}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K})$ in which adversary $\mathcal{A}$ plays a security game $\mathtt{SEC}$, we denote $\mathsf{Succ}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K}) = 1]$ the success probability of this adversary. We additionally denote $\mathsf{Succ}^{\mathsf{sec}}(t, \mathfrak{K}) = \max_{\mathcal{A} \leq t}\{\mathsf{Succ}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K})\}$, the maximal success any adversary running within time t can get.

When one considers a pair of experiments $\mathsf{Exp}^{\mathsf{sec}\text{-}b}(\mathcal{A}, \mathfrak{K})$, for $b = 0, 1$, in which adversary $\mathcal{A}$ plays a security game $\mathtt{SEC}$, we denote

$$\mathsf{Adv}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K}) = \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{sec}-0}(\mathfrak{K}) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{sec}-1}(\mathfrak{K}) = 1\right],$$

the advantage of this adversary. We additionally denote $\mathsf{Adv}^{\mathsf{sec}}(t, \mathfrak{K}) = \max_{\mathcal{A} \leq t}\{\mathsf{Adv}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K})\}$, the maximal advantage any adversary running within time t can get. In an equivalent way, we can consider the experiment $\mathsf{Exp}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K})$ where we first choose a random bit b and then run experiment $\mathsf{Exp}^{\mathsf{sec}\text{-}b}(\mathcal{A}, \mathfrak{K})$. Then the advantage is

$$\mathsf{Adv}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K}) = 2 \cdot \Pr[\mathsf{Exp}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K}) = b] - 1.$$

A.2 Formal Definition of the Primitives

Hash Function Family. A hash function family $(\mathcal{HF}_{\mathfrak{K}})_{\mathfrak{K}}$ is an ensemble (indexed by $\mathfrak{K}$, the security parameter) of families of functions $\mathcal{H}$ from $\{0,1\}^*$ to a fixed-length output, either $\{0,1\}^k$ or $\mathbb{Z}_p$. Such a family is said *collision-resistant* if any polynomial-time adversary $\mathcal{A}$ on a random function $\mathcal{H} \xleftarrow{\$} \mathcal{HF}$ cannot find a collision with non-negligible probability (on $\mathfrak{K}$). More precisely, we denote

$$\mathsf{Succ}^{\mathsf{coll}}(\mathcal{A}, \mathfrak{K}) = \Pr\left[\mathcal{H} \xleftarrow{\$} \mathcal{HF}_{\mathfrak{K}}, (m_0, m_1) \leftarrow \mathcal{A}(\mathcal{H}) : \mathcal{H}(m_0) = \mathcal{H}(m_1)\right].$$

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

- $\mathsf{Setupe}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KGe}(\mathsf{param})$ 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 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 $\perp$.

The first algorithm is often forgotten, to simplify notations. An encryption scheme $\mathcal{E}$ should satisfy the following properties

– *Correctness*: for all key pairs $(\mathsf{ek}, \mathsf{dk})$, all labels 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 (IND-CCA) can be formalized by the security game in Fig. 5, where the adversary $\mathcal{A}$ keeps some internal state between the various calls `FIND` and `GUESS`, and makes use of the oracle ODecrypt:
  - $\mathsf{ODecrypt}(\ell, C)$: 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 `CTXT`.

$\mathsf{Exp}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}-b}(\mathcal{A}, \mathfrak{K})$
  $\mathsf{param} \leftarrow \mathsf{Setupe}(1^{\mathfrak{K}})$
  $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KGe}(\mathsf{param})$
  $(\ell^*, m_0, m_1) \leftarrow \mathcal{A}(\mathtt{FIND} : \mathsf{ek}, \mathsf{ODecrypt}(\cdot, \cdot))$
  $C^* \leftarrow \mathsf{Encrypt}(\ell^*, \mathsf{ek}, m_b)$
  $b' \leftarrow \mathcal{A}(\mathtt{GUESS} : C^*, \mathsf{ODecrypt}(\cdot, \cdot))$
  if $(\ell^*, C^*) \in \mathtt{CTXT}$ **then return** 0
  else return b'

Fig. 5. Experiments $\mathsf{Exp}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}-n}$ for the IND-CCA security

The advantages are

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}}(\mathcal{A}) &= \Pr[\mathsf{Exp}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}-1}(\mathcal{A}, \mathfrak{K}) = 1] - \Pr[\mathsf{Exp}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}-0}(\mathcal{A}, \mathfrak{K}) = 1] \\
\mathsf{Adv}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}}(t, q_d) &= \max_{\mathcal{A} \leq t, q_d} \{\mathsf{Adv}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}}(\mathcal{A})\},
\end{aligned}$$

where we bound the adversaries to work within time t and to ask at most q_d decryption queries.

In some cases, *indistinguishability under chosen-plaintext attacks* (IND-CPA) is enough. This notion is similar to the above IND-CCA except that the adversary has no decryption-oracle ODecrypt access:

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{E}}^{\mathtt{ind\text{-}cpa}}(\mathcal{A}) &= \Pr[\mathsf{Exp}_{\mathcal{E}}^{\mathtt{ind\text{-}cpa}-1}(\mathcal{A}, \mathfrak{K}) = 1] - \Pr[\mathsf{Exp}_{\mathcal{E}}^{\mathtt{ind\text{-}cpa}-0}(\mathcal{A}, \mathfrak{K}) = 1] \\
\mathsf{Adv}_{\mathcal{E}}^{\mathtt{ind\text{-}cpa}}(t) &= \max_{\mathcal{A} \leq t} \{\mathsf{Adv}_{\mathcal{E}}^{\mathtt{ind\text{-}cpa}}(\mathcal{A})\},
\end{aligned}$$

where we bound the adversaries to work within time t: $\mathsf{Adv}_{\mathcal{E}}^{\mathtt{ind\text{-}cpa}}(t) = \mathsf{Adv}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}}(t, 0)$.

A.3 Concrete Instantiations

IND-CPA Encryption: ElGamal. The ElGamal encryption scheme [ElG84] is defined as follows:

– $\mathsf{Setupe}(1^{\mathfrak{K}})$ generates a group $\mathbb{G}$ of order p, with a generator g;
– $\mathsf{KGe}(\mathsf{param})$ generates $\mathsf{dk} = z \overset{\$}{\leftarrow} \mathbb{Z}_p$, and sets, $\mathsf{ek} = h = g^z$;
– $\mathsf{Encrypt}(\mathsf{ek}, M; r)$, for a message $M \in \mathbb{G}$ and a random scalar $r \in \mathbb{Z}_p$, the ciphertext is $C = (u = g^r, v = M \cdot h^r)$;
– $\mathsf{Decrypt}(\mathsf{dk}, C)$: one computes $M = v/u^z$ and outputs M.

This scheme is indistinguishable against chosen-plaintext attacks (IND-CPA), under the DDH assumption.

IND-CCA Encryption: Cramer-Shoup (CS). The Cramer-Shoup encryption scheme [CS98] 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;
– $\mathsf{KGe}(\mathsf{param})$ 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 $\mathcal{H}$ in a hash family $\mathcal{HF}$ (or simply a Universal One-Way Hash Function). The encryption key is $\mathsf{ek} = (g_1, g_2, c, d, h, \mathcal{H})$;
– $\mathsf{Encrypt}(\ell, \mathsf{ek}, M; r)$, for a message $M \in \mathbb{G}$ and a random scalar $r \in \mathbb{Z}_p$, the ciphertext is $C = (\ell, u_1 = g_1^r, u_2 = g_2^r, v = M \cdot h^r, w = (cd^\xi)^r)$, where v is computed after $\xi = \mathcal{H}((\ell, u_1, u_2, v))$.
– $\mathsf{Decrypt}(\ell, \mathsf{dk}, C)$: one first computes $\xi = \mathcal{H}((\ell, u_1, u_2, v))$ 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 $M = v/u_1^z$ and outputs M. Otherwise, one outputs $\perp$.

This scheme is indistinguishable against chosen-ciphertext attacks ($\mathsf{IND\text{-}CCA}$), under the DDH assumption and if one uses a collision-resistant hash function $\mathcal{H}$.

IND-CCA Encryption under DLin: Linear Cramer-Shoup. The Linear Cramer-Shoup encryption scheme [Sha07] is a variant of Cramer-Shoup $\mathsf{IND\text{-}CCA}$ under DLin, instead of DDH. So this scheme can be used with symmetric pairings. Here is the scheme:

– $\mathsf{Setup}(1^k)$ generates a group $\mathbb{G}$ of order p, with three independent generators $(g_1, g_2, g_3) \stackrel{\$}{\leftarrow} \mathbb{G}^3$;
– $\mathsf{KGe}(\mathsf{param})$ generates $\mathsf{dk} = (x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3) \stackrel{\$}{\leftarrow} \mathbb{Z}_p^9$, and sets, for $i = 1, 2$, $c_i = g_i^{x_i} g_3^{x_3}$, $d_i = g_i^{y_i} g_3^{y_3}$, and $h_i = g_i^{z_i} g_3^{z_3}$. It also chooses a hash function $\mathcal{H}$ in a collision-resistant hash family $\mathcal{HF}$ (or simply a Universal One-Way Hash Function). The encryption key is $\mathsf{ek} = (c_1, c_2, d_1, d_2, h_1, h_2, \mathcal{H})$.
– $\mathsf{Encrypt}(\ell, \mathsf{ek}, M; r, s)$, for a message $M \in \mathbb{G}$ and two random scalars $r, s \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, the ciphertext is $C = (u_1 = g_1^r, u_2 = g_2^s, u_3 = g_3^{r+s}, v = M \cdot h_1^r h_2^s, w = (c_1 d_1^\xi)^r (c_2 d_2^\xi)^s)$, where v is computed afterwards with $\xi = \mathcal{H}((\ell, u_1, u_2, u_3, v))$.
– $\mathsf{Decrypt}(\ell, \mathsf{dk}, C = (u_1, u_2, u_3, v, w))$: one first computes $\xi = \mathcal{H}((\ell, u_1, u_2, u_3, v))$ and checks whether $u_1^{x_1+\xi y_1} \cdot u_2^{x_2+\xi y_2} \cdot u_3^{x_3+\xi y_3} \stackrel{?}{=} v$. If the equality holds, one computes $M = v/(u_1^{z_1} u_2^{z_2} u_3^{z_3})$ and outputs M. Otherwise, one outputs $\perp$.

This scheme is indistinguishable against chosen-ciphertext attacks, under the DLin assumption and if one uses a collision-resistant hash function $\mathcal{H}$.

B Proof of Mixed-Randomness

We suppose that $\mathcal{T}_{\mathsf{crs}_1}$ contains enough information to be able to compute the discrete logarithm of elements of $\Gamma^{(1)}$, denoted $\mathfrak{L}(\Gamma^{(1)})$.

Let $m = n_1 - \dim \hat{\mathscr{L}}_1$. First, let us remark that using hybrid methods on pseudo-randomness of $\mathscr{L}_2$, we can prove that the two following distributions of the tuple $U = (C_2, (\mathsf{hp}_{2,j}, H_{2,j})_{1 \le j \le m})$ are computationally indistinguishable:

– normal distribution: $C_2 \stackrel{\$}{\leftarrow} \mathscr{L}_2$ and for each j, $\mathsf{hp}_{2,j}$ is a valid projection key $\boldsymbol{\gamma}^{(2,j)} = (\gamma_l^{(2,j)})_{l=1,\dots,k_2}$ corresponding to some hashing key $\mathsf{hk}_{2,j} = \boldsymbol{\alpha}^{(2,j)} \in \mathbb{Z}_p^{n_2}$, and $H_{2,j}$ is the hash value of C_2 under $\mathsf{hk}_{2,j}$;
– random distribution: C_2, $\mathsf{hp}_{2,j}$ and $\mathsf{hk}_{2,j}$ are defined the same way, but the values $H_{2,j}$ are picked independently and uniformly at random.

The whole proof consists in constructing a valid projection key $\mathsf{hp} = (\gamma_j)_{j=1,\dots,k_1n_2+k_2n_1}$ and a hash value H, given $(\mathsf{hp}_{2,j}, H_{2,j})_{1\le j\le m}$, so that, if the previous tuple U comes from the normal distribution, H is a valid hash value, while otherwise it is uniformly random.

Let $\Delta \in \mathbb{Z}_p^{n_1\times m}$ be a matrix such that the solutions of the linear equation $\mathfrak{L}(\Gamma^{(1)}) \odot \boldsymbol{X} = 0$ (with unknown $\boldsymbol{X} \in \mathbb{Z}_p^{n_1}$) are exactly the vectors $\Delta \odot \boldsymbol{\delta}$ for $\boldsymbol{\delta} \in \mathbb{Z}_p^m$. In other words columns of Δ form a basis of the kernel of $\mathfrak{L}(\Gamma^{(1)})$. Δ can be obtained by doing a Gaussian elimination over $\mathfrak{L}(\Gamma^{(1)})$.

We set the column vector $\boldsymbol{\gamma'^{(2)}} \in \mathfrak{G}_2^{mk_2}$ to be the concatenation of the vectors $\boldsymbol{\gamma^{(2,j)}}$, the column vector $\boldsymbol{\alpha'^{(2)}} \in \mathbb{Z}_p^{mn_2}$ to be the concatenation of $\boldsymbol{\alpha^{(2,j)}}$ (for $j = 1,\dots,m$), and the column vector $\boldsymbol{H_2} \in \mathfrak{G}_2^m$ to be $\boldsymbol{H_2} = (H_{2,j})_{j=1,\dots,m}$. We then have:

$$\boldsymbol{\gamma}'^{\boldsymbol{(2)}} = (\mathsf{Id}_m \otimes \Gamma^{(2)}) \odot \boldsymbol{\alpha}'^{\boldsymbol{(2)}}, \tag{4}$$

and, if U is from the normal distribution:

$$\boldsymbol{H_2} = (\mathsf{Id}_m \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\alpha'^{(2)}}, \tag{5}$$

and otherwise it is random, and we can write it as:

$$\boldsymbol{H_2} = (\mathsf{Id}_m \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\alpha''^{(2)}}, \tag{6}$$

with $\boldsymbol{\alpha''^{(2)}}$ a random vector in $\mathbb{Z}_p^m$ (independent of $\boldsymbol{\alpha'^{(2)}}$).

We then pick a random column vector $\boldsymbol{\varepsilon} \in \mathbb{Z}_p^{n_1n_2}$, and we set:

$$\boldsymbol{\gamma^{(1)}} := (\Gamma^{(1)} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\varepsilon} \tag{7}$$

$$\boldsymbol{\gamma^{(2)}} := (\Delta \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\gamma}'^{\boldsymbol{(2)}} \oplus (\mathsf{Id}_{n_1} \otimes \Gamma^{(2)}) \odot \boldsymbol{\varepsilon} \tag{8}$$

$$H := \hat{\boldsymbol{C}}_1 \odot \Delta \odot \boldsymbol{H_2} \oplus (\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\varepsilon} \tag{9}$$

Let us now prove that $\mathsf{hp} := (\boldsymbol{\gamma^{(1)}}, \boldsymbol{\gamma^{(2)}})$ is a valid projection key for some random vector $\boldsymbol{\alpha}$, and that H is the correct hash value of (C_1, C_2) if the tuple U is distributed normally, and a random value otherwise.

For that purpose, let us set:

$$\boldsymbol{\alpha} := (\Delta \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\alpha'^{(2)}} \oplus \boldsymbol{\varepsilon}. \tag{10}$$

This vector is uniformly random due do $\boldsymbol{\varepsilon}$. In addition, from Equation (7), we get:

$$(\Gamma^{(1)} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\alpha} = ((\Gamma^{(1)} \odot \Delta) \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\alpha'^{(2)}} \oplus (\Gamma^{(1)} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\varepsilon} = \boldsymbol{\gamma^{(1)}},$$

since $\Gamma^{(1)} \odot \Delta = 0$ by definition of Δ. So $\boldsymbol{\gamma^{(1)}}$ is the correct first part of the projection key associated to $\boldsymbol{\alpha}$. And, from Equation (8) and Equation (4), we get:

$$(\mathsf{Id}_{n_1} \otimes \Gamma^{(2)}) \odot \boldsymbol{\alpha} = (\Delta \otimes \Gamma^{(2)}) \odot \boldsymbol{\alpha'^{(2)}} \oplus (\mathsf{Id}_{n_1} \otimes \Gamma^{(2)}) \odot \boldsymbol{\varepsilon} = \boldsymbol{\gamma^{(2)}},$$

because

$$(\Delta \otimes \Gamma^{(2)}) \odot \boldsymbol{\alpha'^{(2)}} = (\Delta \otimes \mathsf{Id}_{n_2}) \odot (\mathsf{Id}_m \otimes \Gamma^{(2)}) \odot \boldsymbol{\alpha'^{(2)}} = (\Delta \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\gamma'^{(2)}},$$

so that $\boldsymbol{\gamma^{(2)}}$ is the correct second part of the projection key associated to $\boldsymbol{\alpha}$, and hp is the projection key associated to $\boldsymbol{\alpha}$.

- If U is from the normal distribution, Equation (5) and Equation (9), we get:

$$\begin{aligned} H &= \hat{\boldsymbol{C}}_1 \odot \Delta \odot (\mathsf{Id}_m \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\alpha'^{(2)}} \oplus (\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\varepsilon} \\ &= ((\hat{\boldsymbol{C}}_1 \odot \Delta) \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\alpha'^{(2)}} \oplus (\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\varepsilon} \\ &= (\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2) \odot (\Delta \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\alpha'^{(2)}} \oplus (\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\varepsilon}. \end{aligned}$$

and so by definition of $\boldsymbol{\alpha}$ (Equation (10)), we have:

$$H = (\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\alpha},$$

hence H is the hash value of (C_1, C_2) under the hashing key of $\boldsymbol{\alpha}$. In this case, everything has been generated as in the mixed pseudo-randomness experiment $\mathsf{Exp}^{\texttt{mixed-ps-rnd-}b}$ (Fig. 4) for $b = 0$.

– if U is from the random distribution, as previously, from Equation (6) and Equation (9), we get that:

$$H = (\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\alpha}',$$

where

$$\boldsymbol{\alpha}' = (\Delta \odot \boldsymbol{\alpha}''^{(2)} \oplus \boldsymbol{\varepsilon}).$$

Since $\boldsymbol{\alpha}''^{(2)}$ is random and independent of everything else, and by definition of Δ, $\boldsymbol{\alpha}'$ can be seen as an independent hashing key chosen uniformly at random among the keys verifying:

$$(\Gamma^{(1)} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\alpha}' = (\Gamma^{(1)} \otimes \mathsf{Id}_{n_2}) \odot \boldsymbol{\varepsilon}.$$

Since $C_1 \notin \mathscr{L}_1$, $\hat{\boldsymbol{C}}_1$ is linearly independent from rows of $\Gamma^{(1)}$, and $\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2$ is linearly independent from rows of $\Gamma^{(1)} \otimes \mathsf{Id}_{n_2}$, hence $H = (\hat{\boldsymbol{C}}_1 \otimes \hat{\boldsymbol{C}}_2) \odot \boldsymbol{\alpha}'$ looks uniformly random. Therefore, in this case, where U is from the random distribution, everything has been generated as in the mixed pseudo-randomness experiment $\mathsf{Exp}^{\texttt{mixed-ps-rnd-}b}$ (Fig. 4) for $b = 1$.

Since the normal distribution of U is computationally indistinguishable from the random one, this proves the mixed pseudo-randomness property.

C Application Details

C.1 Threshold Cramer-Shoup-like Encryption Schemes

In this appendix, we give some details on our constructions of threshold Cramer-Shoup-like encryption schemes. We first give the non-threshold schemes together with an IND-CCA proof and then we show that these schemes can easily be decrypted in a threshold way.

(Non-Threshold) Constructions. Our two encryption schemes in Section 5.6 and Section 8.4 work as follows:

- $\mathsf{Setupe}(1^{\mathfrak{K}})$ generates an asymmetric bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$;
- $\mathsf{KGe}(\mathsf{param})$ generates $\mathsf{crs}_1 = (g_{1,1}, g_{1,2}) \stackrel{\$}{\leftarrow} (\mathbb{G}_1 \setminus \{1\})^2$ together with a CRS σ for a one-time simulation-sound NIZK for the language defined by the following witness relation:

  $$\mathcal{R}_{1,\mathsf{crs}_1}((u_1, u_2), r) = 1 \qquad \text{if and only if} \qquad u_1 = g_{1,1}^r \quad \text{and} \quad u_2 = g_{1,2}^r;$$

  then it chooses $z \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and sets $h \leftarrow g_{1,1}^z$. It also chooses a hash function $\mathcal{H}$ in a collision-resistant hash family $\mathcal{HF}$. The encryption key is $\mathsf{ek} = (g_{1,1}, g_{1,2}, h, \sigma, \mathcal{H})$, while the decryption key is $\mathsf{dk} = (z, \sigma, \mathcal{H})$;
- $\mathsf{Encrypt}(\ell, \mathsf{ek}, M; r)$, for a message $M \in \mathbb{G}$ and a random scalar $r \in \mathbb{Z}_p$, outputs the following ciphertext $C = (\ell, u_1 = g_{1,1}^r, u_2 = g_{1,2}^r, v = M \cdot h^r, \boldsymbol{\pi} = \mathsf{Prove}(\sigma, (g_{1,1}, g_{1,2}), \xi, (u_1, u_2), r)$, where $\xi = \mathcal{H}((\ell, u_1, u_2, v))$.
- $\mathsf{Decrypt}(\ell, \mathsf{dk}, C)$ first computes $\xi = \mathcal{H}((\ell, u_1, u_2, v))$ and checks whether the proof π is valid $(\mathsf{Ver}(\sigma, (g_{1,1}, g_{1,2}), \xi, (u_1, u_2), \boldsymbol{\pi}) \stackrel{?}{=} 1)$. If the equality holds, it computes $M = v/u_1^z$ and outputs M. Otherwise, it outputs $\perp$.

First Construction. Here is the concrete first construction of one-time simulation-sound NIZK, following the construction in Section 5.4:

– $\mathsf{Setup}(\mathsf{crs}_1)$ picks a random group element $h_2 \in \mathbb{G}_2$ and a random vector $\boldsymbol{\alpha} \in \mathbb{Z}_p^8$ and sets:

$$\begin{aligned}
\Gamma^{(1)} &:= \begin{pmatrix} g_{1,1} & g_{1,2} & 1 & 1 \\ 1 & 1 & g_{1,1} & g_{1,2} \end{pmatrix} \\
\Gamma^{(2)} &:= \begin{pmatrix} g_2 & h_2 \end{pmatrix} \\
\boldsymbol{\gamma}^{(1)} &:= \left(\Gamma^{(1)} \otimes \mathsf{Id}_2\right) \odot \boldsymbol{\alpha} = \begin{pmatrix} g_{1,1} & 1 & g_{1,2} & 1 & 1 & 1 & 1 & 1 \\ 1 & g_{1,1} & 1 & g_{1,2} & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & g_{1,1} & 1 & g_{1,2} & 1 \\ 1 & 1 & 1 & 1 & 1 & g_{1,1} & 1 & g_{1,2} \end{pmatrix} \odot \boldsymbol{\alpha} \\
\boldsymbol{\gamma}^{(2)} &:= \left(\mathsf{Id}_4 \otimes \Gamma^{(2)}\right) \odot \boldsymbol{\alpha} = \begin{pmatrix} g_2 & h_2 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & g_2 & h_2 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & g_2 & h_2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & g_2 & h_2 \end{pmatrix} \odot \boldsymbol{\alpha};
\end{aligned}$$

  (where 1 is the group element $1 \in \mathbb{G}_1$). The CRS is $\sigma := (\boldsymbol{\gamma}^{(1)}, \boldsymbol{\gamma}^{(2)})$, while the trapdoor is $\mathcal{T} = \boldsymbol{\alpha}$.

– $\mathsf{Prove}(\sigma, \mathsf{crs}_1, \xi, (u_1, u_2), r)$ just sets $\mathsf{tag} = \xi$ and outputs the proof[4]

$$\boldsymbol{\pi} \leftarrow \left(\begin{pmatrix} r & r\xi \end{pmatrix} \otimes \mathsf{Id}_2\right) \odot \boldsymbol{\gamma}^{(2)} = \begin{pmatrix} r & 0 & r\xi & 0 \\ 0 & r & 0 & r\xi \end{pmatrix} \odot \boldsymbol{\gamma}^{(1)}, \tag{11}$$

  which is a column vector of two elements in $\mathbb{G}_1$.

– $\mathsf{Ver}(\sigma, \mathsf{crs}_1, \xi, (u_1, u_2), \boldsymbol{\pi})$ checks whether:

$$\Gamma^{(2)} \odot \boldsymbol{\pi} \stackrel{?}{=} \begin{pmatrix} u_1 & u_2 & u_1^\xi & u_2^\xi \end{pmatrix} \odot \boldsymbol{\gamma}^{(2)}.$$

If we know the trapdoor $\mathcal{T}$ of the NIZK, namely the hashing key $\boldsymbol{\alpha}$ of the SPHF, the verification of the NIZK can be performed as follows:

$$\boldsymbol{\pi} \stackrel{?}{=} \left(\begin{pmatrix} u_1 & u_2 & u_1^\xi & u_2^\xi \end{pmatrix} \otimes \mathsf{Id}_2\right) \odot \boldsymbol{\alpha} = \begin{pmatrix} u_1 & 1 & u_2 & 1 & u_1^\xi & 1 & u_2^\xi & 1 \\ 1 & u_1 & 1 & u_2 & 1 & u_1^\xi & 1 & u_2^\xi \end{pmatrix} \odot \boldsymbol{\alpha}. \tag{12}$$

Indeed, Equation (12) is just this equality multiplied by the matrix $\Gamma^{(2)}$, so this verification method will always reject when the original one rejects, and the one-time simulation-soundness still holds. It just remains to check that using this stronger verification method does not break the completeness: the completeness still holds because Equation (11) implies:

$$\begin{aligned}
\boldsymbol{\pi} &= \left(\begin{pmatrix} r & r\xi \end{pmatrix} \otimes \mathsf{Id}_2\right) \odot (\Gamma^{(1)} \otimes \mathsf{Id}_2) \odot \boldsymbol{\alpha} = \left(\begin{pmatrix} r & r\xi \end{pmatrix} \odot \Gamma^{(1)}\right) \otimes (\mathsf{Id}_2 \odot \mathsf{Id}_2) \odot \boldsymbol{\alpha} \\
&= \left(\begin{pmatrix} u_1 & u_2 & u_1^\xi & u_2^\xi \end{pmatrix} \otimes \mathsf{Id}_2\right) \odot \boldsymbol{\alpha}.
\end{aligned}$$

Second Construction. The second construction follows the construction in Section 8.2. It is very similar to the first one. The only difference is that the tag $\mathsf{tag} = \xi$ is used bit by bit, instead of all at once, and that the matrix $\Gamma^{(1)}$ can be seen as a block diagonal matrix with $\nu = |\xi|$ blocks equal to the above matrix $\Gamma^{(1)}$. Moreover, we remark that, as for the first construction, knowing the trapdoor $\mathcal{T}$ of the NIZK enables to decrypt without performing any pairing computations.

[4] In the original construction tag would be the hash value of ξ and (u_1, u_2) under some collision-resistant hash function, but here ξ already "contains" (u_1, u_2) so we can slightly simplify the construction by choosing $\mathsf{tag} = \xi$.

IND-CCA Security Proof. The proof is quite straightforward and basically uses ideas in the security proof of the Cramer-Shoup encryption scheme [CS98]. Here is a sketch of a sequence of games proving the IND-CCA property:

Game $\mathbf{G}_0$: This is the game for $\mathsf{Exp}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}-b}$ for $b = 0$ (see Section A.2).

Game $\mathbf{G}_1$: In this game, we generate $g_{1,2}$ as $g_{1,1}^t$ (with $t \xleftarrow{\$} \mathbb{Z}_p$) and reject all ciphertexts $C = (u_1, u_2, v, \boldsymbol{\pi})$ submitted to the decryption oracle for which $u_2 \neq u_2^t$. This game is indistinguishable from the previous one under the soundness of the NIZK, which ensures that if the proof $\boldsymbol{\pi}$ is not rejected, $(g_{1,1}, g_{1,2}, u_1, u_2)$ is a DDH tuple.

Game $\mathbf{G}_2$: In this game, we generate h as $h = g_{1,1}^{z_1} g_{1,2}^{z_2}$ (with $z_1, z_2 \xleftarrow{\$} \mathbb{Z}_p$) instead of $h = g_{1,1}^z$. In addition, we decrypt ciphertexts $C = (u_1, u_2, v, \boldsymbol{\pi})$ by first rejecting if $\boldsymbol{\pi}$ is not a valid proof or $u_2 \neq u_1^t$ (as before) and then outputting $v/(u_1^{z_1} u_2^{z_2})$ (instead of v/u_1^z). This game is perfectly indistinguishable from the previous one, because h can be written $h = g_{1,1}^{z_1+tz_2}$ and $u_1^{z_1} u_2^{z_2} = u_1^{z_1+tz_2}$.

Game $\mathbf{G}_3$: In this game, we do not check anymore that $u_2 = u_1^t$ and we generate $g_{1,2}$ directly as a random group element in $\mathbb{G}_1$. This game is indistinguishable from the previous one under the soundness of the NIZK.

Game $\mathbf{G}_4$: In this game, for the challenge ciphertext $C^* = (u_1^*, u_2^*, v^*, \boldsymbol{\pi}^*)$, we compute v^* as $v^* = u_1^{*z_1} u_2^{*z_2}$, instead of h^r, where $u_1 = g_{1,1}^r$ and $u_2 = g_{1,2}^r$. This game is perfectly indistinguishable from the previous one.

Game $\mathbf{G}_5$: In this game, we simulate the proof $\boldsymbol{\pi}^*$ in the challenge ciphertext $C^* = (u_1^*, u_2^*, v^*, \boldsymbol{\pi}^*)$. This game is indistinguishable from the previous one under the zero-knowledge property of the NIZK. In addition, in this game, knowledge of r in C^* is no longer required.

Game $\mathbf{G}_6$: In this game, we replace (u_1^*, u_2^*) which were a DDH tuple in basis $(g_{1,1}, g_{1,2})$ by a random tuple. This game is indistinguishable from the previous one under the DDH assumption.

Game $\mathbf{G}_7$: In this game, we again generate $g_{1,2}$ as $g_{1,1}^t$ (with $t \xleftarrow{\$} \mathbb{Z}_p$) and reject all ciphertexts $C = (u_1, u_2, v, \boldsymbol{\pi})$ submitted to the decryption oracle for which $u_2 \neq u_2^t$. This game is insdinstiguishable form the previous one under the one-time simulation-soundness of the NIZK.

Game $\mathbf{G}_8$: As in Cramer-Shoup's proof [CS98], it is easy to show that the only information (from an information theoretic point of view) the adversary sees of z_1 and z_2 except from C^* is $z_1 + tz_2$. So $u_1^{*z_1} u_2^{*z_2}$ looks completely random to the adversary if (u_1, u_2) is not a DDH tuple in basis $(g_{1,1}, g_{1,2})$ (which happens with probability $1 - 1/p$). Therefore we can replace v^* by a random value, and this game is statistically indistinguishable from the previous one. Finally, we can redo all the previous games in the reverse order and sees that $\mathsf{Exp}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}-b}$ with $b = 0$ is indistinguishable from $\mathsf{Exp}_{\mathcal{E}}^{\mathtt{ind\text{-}cca}-b}$ with $b = 1$.

Threshold Version. The validity of the ciphertext can be verified publicly, just knowing ek (or more precisely σ), and not dk, and then after this test has been performed, we just need to compute v/u_1^z, to get the message. We often say in this case that the ciphertext is "publicly verifiable", though it is not clear that a proper definition exists.

In any case, this property just means that to threshold decrypt the ciphertext, we just need to use Shamir's threshold secret sharing over $\mathbb{Z}_p$ [Sha79], exactly as in [SG02]. If in addition, we want to be able to verify decryption shares without random oracle, we can replace the Fiat-Shamir-based NIZK in [SG02] by one of ours in Section 5.2.

C.2 One-Round Group Password Authenticated Key Exchange

In this appendix, we give some details on our one-round group password authenticated key exchange (GPAKE). We first recall the formal model, then show our construction and finally prove its security.

Formal Model. Let us recall the model of group password authenticated key exchange for n users in [ABCP06].

We denote by $U_1, \ldots, U_n$ the parties that can participate in the key exchange protocol P. Each of them may have several *instances* called oracles involved in distinct, possibly concurrent, executions of P. We denote U_i instances by $\Pi_{U_i}^s$. To simplify notations, s is supposed to be an integer between 1 and $q_{\mathtt{session}}$, and for any s, instances $(\Pi_{U_i}^s)_{i=1,\ldots,n}$ are supposed to run the protocol together. Each s therefore corresponds to a session of the protocol, and is called a session id. The parties share a low-entropy secret pw^* which is uniformly drawn from a small dictionary $\mathsf{Password}$ of size N.

The key exchange algorithm P is an interactive protocol between the U_i's that provides the instances with a session key sk. During the execution of this protocol, the adversary has the entire control of the network, and tries to break the privacy of the key.

As in [ABCP06], we use the Real-or-Random notion for the semantic security instead of the Find-then-Guess. This notion is strictly stronger in the password-based setting. And actually, since we focus on the semantic security only, we can assume that each time a player accepts a key, the latter is revealed to the adversary, either in a real way, or in a random one (according to a bit b). Let us briefly review each query:

- $\mathtt{Send}(U_i, s, U_j, m)$: This query enables to consider active attacks by having $\mathcal{A}$ sending a message to the instance $\Pi_{U_i}^s$ in the name of U_j. The adversary $\mathcal{A}$ gets back the response $\Pi_{U_i}^s$ generates in processing the message m according to the protocol P. A special query $\mathtt{Send}(U_i, s, \mathtt{Start})$ initializes the instance $\Pi_{U_i}^s$ and the key exchange algorithm, and thus the adversary receives the initial flows sent out by the instance.
- $\mathtt{Test}^b(U_i, s)$: This query models the misuse of the session key by instance $\Pi_{U_i}^s$ (*known-key attacks*). The query is only available to $\mathcal{A}$ if the attacked instance actually "holds" a session key. It either releases the actual key to $\mathcal{A}$, if $b = 1$ or a random one, if $b = 0$. The random keys must however be consistent between users in the same session. Therefore, a random key is simulated by the evaluation of a random function on the view a user has of the session: all the partners have the same view, they thus have the same random key (but independent of the actual view.)

  Remark 8. Note that it has been shown [AFP05] that this query is indeed enough to model *known-key attacks* —where $\mathtt{Reveal}$ queries, which always answer with the real keys, are available—, and makes the model even stronger. Even though their result has only been proven in the two-party and three-party scenarios, one should note that their proof can be easily extended to the group scenario.

As already noticed, the aim of the adversary is to break the privacy of the session key (a.k.a., semantic security). This security notion takes place in the context of executing P in the presence of the adversary $\mathcal{A}$. One first draws a password pw^* from $\mathsf{Password}$, flips a coin b, provides coin tosses to $\mathcal{A}$, as well as access to the $\mathtt{Test}^b$ and $\mathtt{Send}$ oracles.

The goal of the adversary is to guess the bit b involved in the $\mathtt{Test}$ queries, by outputting this guess b'. We denote the **AKE advantage** as the probability that $\mathcal{A}$ correctly guesses the value of b. More precisely we define $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ake}}(\mathfrak{K}) = 2\Pr[b = b'] - 1$. The protocol P is said to be (t, ϵ)-*AKE-secure* if $\mathcal{A}$'s advantage is smaller than ϵ for any adversary $\mathcal{A}$ running with time t.

We will denote by $q_{\mathtt{active}}$ the number of messages the adversary produced by himself (thus without including those he has just forwarded). This number upper-bounds the number of on-line "tests" the adversary performs to guess the password. And we denote by $q_{\mathtt{session}}$ the total number of sessions the adversary has initiated: $nq_{\mathtt{session}}$, where n is the size of the group, upper-bounds the total number of messages the adversary has sent in the protocol (including those he has built and those he has just forwarded).

The best we can expect with such a scheme is that the adversary erases no more than 1 password for each *session* in which he plays actively (since there exists attacks which achieve that in any password-based scheme). However, in our scheme, we can just prevent the adversary from erasing more than 1 password for each *player* that he tries to impersonate, which was also the case for the scheme in [ABCP06]. So we want to prove that $\mathsf{Adv}^{\mathsf{ake}}_{\mathcal{A}}(\mathfrak{K})$ is bounded by $q_{\mathtt{active}}/N$ plus some negligible term in $\mathfrak{K}$.

Finally, we are interested in one-round protocol, meaning that each player sends exactly one flow and all flows can be sent simultaneously. Since the communication channel is not assumed to be reliable, the adversary is allowed to modify messages, to delete them, and to alter the order in which these messages are received.

Protocol. The protocol is the one described in Section 6.1 with this slight modification to take into account the session id s: an instance $\Pi^s_{U_i}$ also sends the session id s and concatenate s to all ciphertexts labels. In addition $\Pi^s_{U_i}$ refuses messages not for the session id s.

SPHF for Linear Cramer-Shoup Ciphertexts. Our construction in Section 6.1 needs an SPHF for the language defined by the following witness relation:

$$\mathcal{R}_{\mathsf{par}}((\ell, C), r) \qquad \text{if and only if} \qquad C = \mathsf{Encrypt}(\ell, \mathsf{ek}, \mathsf{par}; r)$$

where Encrypt is the encryption algorithm for Linear Cramer-Shoup, ek is an encryption key (stored in crs) and par will be the password we use in our application.

Here is a diverse vector space for the above language (using notations in Section A.2 for ek):

$$\begin{aligned}\hat{\mathcal{X}} &= \mathbb{G}^7 \\ \Gamma &= \begin{pmatrix} g_1 & 1 & 1 & 1 & g_3 & h_1 & c_1 \\ 1 & g_1 & 1 & 1 & 1 & 1 & d_1 \\ 1 & 1 & g_2 & 1 & g_3 & h_2 & c_2 \\ 1 & 1 & 1 & g_2 & 1 & 1 & d_2 \end{pmatrix} \\ \theta_{\mathsf{par}}((\ell, C)) &= (u_1, u_1^\xi, u_2, u_2^\xi, u_3, v, w),\end{aligned}$$

with $C = (u_1, u_2, u_3, v, w)$ and $\xi = \mathcal{H}((\ell, u_1, u_2, u_3, v))$. It is a straightforward extension of the one for Cramer-Shoup encryption scheme introduced in [BBC+13].

Security Proof. The security proof is a delicate extension of the proof for the one-round PAKE of Katz and Vaikuntanathan in [KV11]. It also works for our extension of the protocol for n players (and not only for our protocol for $n = 3$ players), as long as n is logarithmic in $\mathfrak{K}$.

Although our proof is self-contained, we highly recommend the reader to get familiar with the work of Katz and Vaikunthanatan [KV11] before reading this proof. The following paragraphs explain the main difference between our proof and their proof.

Main Difficulties in the Proof. Basically, the main difficulty in the proof comes from the fact that we have to be able to prove that the hash values computed by the honest user look random if the adversary only generated ciphertexts not containing the valid password pw^*. In the case of a two-player PAKE, this was handled by the technical lemma in [KV11]. Its proof basically consisted in an hybrid over all pairs of honestly generated hashing/projection key $(\mathsf{hk}, \mathsf{hp})$ and an honestly generated ciphertext C. The hash value of C under hk can be proven to look random under the pseudo-randomness of the SPHF, which comes from the hard subset membership property of the underlying language (which itself comes from the IND-CCA property of the encryption scheme). More precisely, we could just replace C by an encryption of a dummy password, and that would prove that the hash value of C under hp looks random to someone not knowing hk. This is possible thanks to the IND-CCA property and the fact we do not need to use the random coins of C in that part of the game: these random coins are indeed only used to compute the resulting secret keys, but either these secret keys were already random (if one of the user involved in the session yielding the secret key is corrupted by the adversary and did not used the valid password), or these secret keys can be computed directly using the hashing keys of honest users, without requiring to know the random coins of C. That is why the hash value of C under hk can be replaced by a random value, and then we can change back C to a valid ciphertext of pw^* (and continue the hybrid argument...).

Unfortunately, in our case, we cannot simply do that, since hash values are now over $n-1$ ciphertexts, and as soon as one ciphertext is a valid ciphertext of pw^*, the hash value could be derived from the projection key hp (at least information theoretically). That is why we need to do a much more delicate hybrid over all sets S of possible honest players, and turn all the ciphertexts of these players into ciphertexts of dummy values, assuming there is only one session for each set of players to simplify. However, that needs to be done in the correct order, otherwise, we may not be able to compute the secret keys of the other players by doing so! Basically, what we show is that if we enumerate the sets S by increasing size, everything works.

Another subtlety is that the classical smoothness is not enough, and we need to use the $(n-1)$-smoothness property.

Proof Details. We can assume that there are two kinds of Send-queries: $\mathtt{Send}_0(U_i, s, \mathtt{Start})$ and $\mathtt{Send}_1(U_i, s, U_j, m)$. $\mathtt{Send}_0(U_i, s, \mathtt{Start})$-queries are queries where the adversary asks the instance $\Pi^s_{U_i}$ to send its flow. It is answered by the flow U_i should send to all the U_j with $j \neq i$. $\mathtt{Send}_1(U_i, s, U_j, m)$-queries are queries where the adversary sends the message m to the instance $\Pi^s_{U_i}$. It gives no answer back, but, it may define the session key, for possible later $\mathtt{Test}$-queries, when $\Pi^s_{U_i}$ received a flow from all the other users (U_k with $k \neq i, j$).

We write $\mathsf{Adv}_I(\mathcal{A})$ the advantage of $\mathcal{A}$ in Game $\mathbf{G}_I$ and negl() means negligible in $\mathfrak{K}$.

Game $\mathbf{G}_0$: This game is the real attack game.

Game $\mathbf{G}_1$: We first modify the way one answers the $\mathtt{Send}_1$-queries, by using a decryption oracle, or alternatively knowing the decryption key. More precisely, when a message (hp, C) is sent, two cases can appear:

- it has been generated (altered) by the adversary, then one first decrypts the ciphertext to get the password pw used by the adversary. And, if it is correct ($\mathsf{pw} = \mathsf{pw}^*$) —event $\mathsf{EventStop}$— one declares that $\mathcal{A}$ succeeds (saying that $b' = b$) and terminates the game. Otherwise, we do nothing;
- 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 associated hash values using the hashing key.

The first case can only increase the advantage of the adversary in case $\mathsf{EventStop}$ happens (which probability is computed in $\mathbf{G}_4$). The second change does not affect the way the key is computed, so finally: $\mathsf{Adv}_0(\mathcal{A}) \leq \mathsf{Adv}_1(\mathcal{A}) + \mathrm{negl}(\mathfrak{K})$.

Game $\mathbf{G}_2$: We modify the way the secret keys are computed: each time two simulated instances have corresponding transcripts, the second instance which computes the secret key, does not recompute it but uses the one already computed by the first instance. More precisely, when an instance $\Pi^s_{U_i}$ has received a message (hp_j, C_j) from U_j and previously received messages (hp_k, C_k) from all the other users U_k with $k \neq i, j$, and if another instance $\Pi_{U^t_l}$ received the same messages (hp_k, C_k) for $k \neq l$, and sent (hp_l, C_l), then we set the secret key of $\Pi^s_{U_i}$ to be the one computed by $\Pi^t_{U_l}$. This change is only formal and $\mathsf{Adv}_1(\mathcal{A}) = \mathsf{Adv}_2(\mathcal{A})$.

Game $\mathbf{G}_3$: We modify again the way one answers the $\mathtt{Send}_1$-queries. More precisely, when an instance $\Pi^s_{U_i}$ has received a message (hp_j, C_j) from U_j and previously received messages (hp_k, C_k) from all the other users U_k with $k \neq i, j$, then we choose sk at random. The proof is a non-trivial extension of the technical lemma of Katz and Vaikuntanathan in [KV11].

As explained above, we consider a sequence of hybrid games $\mathbf{G}_{3,h}$, where h is a tuple of the form (s, i, S), where s is a session id ($s \in \{1, \ldots, q_{\mathtt{session}}\}$), i is a player id ($i \in \{1, \ldots, n\}$), and S is a strict subset of $\{1, \ldots, n\}$ which does not contains i. We choose an arbitrary (total)

order $\prec$ over the tuples h so that if $h = (s, i, S) \prec h' = (s', i', S')$, then $|S| \leq |S'|$. We also suppose there exists a special $h = \perp$, which is less ($\prec$) than all regular h tuples. Furthermore, $h - 1$ denotes the tuple h just before h in the order $\prec$. Note there are $q_{\mathtt{session}} n 2^{n-1} + 1$ tuples $h = (s, i, S)$. This number of tuples is much higher than the number of sessions, and the number of hybrid games is exponential on n, because we do not know in advance the structure of the set S for the session s. Hence, we need to enumerate all the possibilities.
We denote by $\mathsf{hk}_{i,s}$ the hashing key honestly generated by $\Pi^s_{U_i}$ (supposing wlog. that any instance always generate such hashing key even if it is not asked by the adversary, through a Send_0 query) and by $\mathsf{hp}_{i,s}$ the associated projection key (which is sent by $\Pi^s_{U_i}$, if asked by the adversary). We also say an hash value we have to compute is of type h, if it is a hash value under $\mathsf{hk}_{i,s}$ of ciphertexts $(C_j)_{j \neq i}$ (with labels $(\ell_j)_{j \neq i}$), where:

- for $j \in S$, C_j was honestly generated by $\Pi^s_{U_j}$,
- for $j \notin S$, C_j was generated (altered) by the adversary (and so we know that these ciphertexts C_j are not valid ciphertexts of pw^*).

Notice there may be up to $|S| + 1$ hash values of type h, since $\Pi^s_{U_i}$ and all the $\Pi^s_{U_j}$ with $j \in S$ may need to compute a hash value of type h. In addition, when $|S| = n - 1$, all these hash values are equal. Therefore, we can see there are at most $n - 1$ distinct hash values of type h, hence the requirement of a $(n - 1)$-smooth SPHF (details follow).
The hybrid $\mathbf{G}_{3,h}$, with $h = (s, i, S)$ is defined as follows: all hash values of types $\preceq h$ are replaced by random values. Clearly $\mathbf{G}_{3,\perp}$ is $\mathbf{G}_2$, while $\mathbf{G}_{3,\top}$ is $\mathbf{G}_3$, where $\top$ is the maximal tuple for $\prec$.
Let $h \neq \perp$ be a tuple (s, i, S). We now just need to prove that $\mathbf{G}_{3,h-1}$ is computationally indistinguishable from $\mathbf{G}_{3,h}$. This is basically done by the following sequence of sub-hybrid games:

Game $\mathbf{G}_{3.0}$: This game is $\mathbf{G}_{3,h-1}$.

Game $\mathbf{G}_{3.1}$: Let $C_{j,s}$ be the ciphertext honestly generated by $\Pi^s_{U_j}$, for $j \in \{1, \dots, n\}$, and $\ell_{j,s}$ be the associated label. In this game, for $j \in S$, when $\Pi^s_{U_j}$ wants to compute the hash value of the ciphertext he received under a projection key hp from some adversarily generated flow from some $\Pi^s_{U_k}$, then:

- either this hash value if of type $\prec h$, in which case, it is actually chosen at random,
- or, it is not, in which case, this implies that at least $|S|$ flows received by $\Pi^s_{U_j}$ are honestly generated. In the previous game, we would compute the requested hash value using the random coins used in $C_{j,s}$, as witness. In this game, we do not want to do that, and instead we remark that among these $|S|$ honest flows, at least one comes from a $\Pi^s_{U_k}$ for $k \notin S$ (otherwise S would contain at least $|S| - 1 + 1$ values, the "+1" coming from the fact $j \in S$. Therefore, we can compute the requested hash values using the random coins of $C_{k,s}$ as witness.

This game is clearly perfectly indistinguishable from the previous one. In addition, in this game, the random coins of $C_{j,s}$ are never used.

Game $\mathbf{G}_{3.2}$: In this game, we now generate $C_{j,s}$ for $j \in S$, as a ciphertext of a dummy (invalid) password. This game is indistinguishable from the previous one, we use the $\mathsf{IND\text{-}CCA}$ property of the encryption scheme (in a classical hybrid way).

Game $\mathbf{G}_{3.3}$: Now, the $(n-1)$-*smoothness* property of the SPHF ensures that the (at most $n-1$) hash values of type h look like independent random values, since these hash values are for words outside the language (all the ciphertexts used are either honestly generated for $C_{j,s}$ for $j \in S$, and so containing a dummy password, or adversarily generated, and so not containing the correct password pw^*). So in this game, we now replace all hash values of type h by independent random values.

Game $\mathbf{G}_{3.4}$: In this game, we encrypt again pw^* in $C_{j,s}$ (for $j \in S$). This game is computationally indistinguishable from the previous one under the $\mathsf{IND\text{-}CCA}$ property of the encryption scheme.

Game $\mathbf{G}_{3.5}$: In this game, we undo what has been done in $\mathbf{G}_{3.1}$. This game is perfectly indistinguishable from the previous one. This game is also exactly $\mathbf{G}_{3,h}$.

The last hybrid $\mathbf{G}_{3,\top}$ corresponds to our current game $\mathbf{G}_3$, i.e. to the case where all secret keys are computed at random. We proved that this game is indistinguishable from the previous and that:

$$|\mathsf{Adv}_3(\mathcal{A}) - \mathsf{Adv}_2(\mathcal{A})| \leq q_{\mathtt{session}} n 2^{n-1} \cdot \mathrm{negl}(),$$

since there are $q_{\mathtt{session}} n 2^{n-1} + 1$ hybrids.

Game $\mathbf{G}_4$: We now modify the way one answers the $\mathtt{Send}_0$-queries: instead of encrypting the correct values, we encrypt a dummy password. Under the $\mathsf{IND\text{-}CCA}$ security of the encryption scheme: $|\mathsf{Adv}_4(\mathcal{A}) - \mathsf{Adv}_3(\mathcal{A})| \leq \mathrm{negl}()$.

If there is no event $\mathsf{EventStop}$ (even $\neg\mathsf{EventStop}$), then this last game looks exactly the same when $b = 0$ and when $b = 1$, hence:

$$\begin{aligned}
\mathsf{Adv}_4(\mathcal{A}) &\leq 2\big(\Pr\left[\, b' = b \mid \neg\mathsf{EventStop} \,\right] \cdot \Pr\left[\, \neg\mathsf{EventStop} \,\right] \\
&\qquad\qquad + \Pr\left[\, b' = b \mid \mathsf{EventStop} \,\right] \cdot \Pr\left[\, \mathsf{EventStop} \,\right]\big) - 1 \\
&\leq 2 \cdot \left(\frac{1}{2}\left(1 - \Pr\left[\, \mathsf{EventStop} \,\right]\right) + \Pr\left[\, \mathsf{EventStop} \,\right]\right) - 1 \\
&\leq \frac{1}{2} + \Pr\left[\, \mathsf{EventStop} \,\right].
\end{aligned}$$

Since in $\mathbf{G}_4$, pw^* is never used before $\mathsf{EventStop}$ happens, and since $\mathsf{EventStop}$ happens when the adversary correctly encrypts pw^* (in one of its $q_{\mathtt{active}}$ active queries), the probability of this event is at most than $q_{\mathtt{active}}/N$. In addition, combining all relations above, we also get:

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathcal{A}}(\mathfrak{K}) = \mathsf{Adv}_0(\mathcal{A}) \leq \mathsf{Adv}_4(\mathcal{A}) + q_{\mathtt{session}} n 2^{n-1} \cdot \mathrm{negl}().$$

Therefore, we have:

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathcal{A}}(\mathfrak{K}) \leq \frac{q_{\mathtt{active}}}{N} + q_{\mathtt{session}} n 2^{n-1} \cdot \mathrm{negl}().$$

That concludes the proof, since $n2^{n-1} \cdot \mathrm{negl}()$ is negligible in $\mathfrak{K}$ when n is logarithmic in $\mathfrak{K}$.