This is the Full Version of the part on Trapdoor Smooth Projective Hash Functions
of the Extended Abstract "New Techniques for SPHFs and Efficient One-Round PAKE Protocols"
that appears in Advances in Cryptology – Proceedings of CRYPTO 2013 (18–22 august 2013, Santa Barbara, California, USA)
R.Canetti and J. A. Garay Eds. Springer-Verlag, LNCS 8042, pages 449–475, http://dx.doi.org/10.1007/978-3-642-40041-4_25.

Trapdoor Smooth Projective Hash Functions

Fabrice Benhamouda and David Pointcheval

ENS, Paris, France *

Abstract. Katz and Vaikuntanathan recently improved smooth projective hash functions in order to build one-round password-authenticated key exchange protocols (PAKE). To achieve security in the UC framework they allowed the simulator to extract the hashing key, which required simulation-sound non-interactive zero-knowledge proofs that are unfortunately inefficient.

We improve the way the latter extractability is obtained by introducing the notion of trapdoor smooth projective hash function (TSPHF). A TSPHF is an SPHF with a trapdoor, which may not allow to recover the complete hashing key, but which still allows to compute the hash value, which is enough for an application to PAKE with UC-security against static corruptions. We additionally show that TSPHFs yield zero-knowledge proofs in two flows, with straight-line extractability.

Besides those quite interesting applications of TSPHF, we also show how to generically build them on languages of ciphertexts, using any ElGamal-like encryption. Our concrete instantiations lead to efficient one-round UC-secure PAKE, extractable zero-knowledge arguments, and verifiable encryption of Waters signatures. In the case of the PAKE, our construction is the most efficient one-round UC-secure PAKE to date.

Keywords. Authenticated Key Exchange, Zero-Knowledge Arguments, Verifiable Encryption, Trapdoor Smooth Projective Hash Functions.

1 Introduction

Smooth Projective Hash Functions (SPHFs) were 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]. Basically, SPHFs are families of pairs of functions $(\mathsf{Hash}, \mathsf{ProjHash})$ defined on a language L. These functions are indexed by a pair of associated keys $(\mathsf{hk}, \mathsf{hp})$, where hk, the hashing key, can be seen as the private key and hp, the projection key, as the public key. On a word $W \in \mathrm{L}$, both functions should lead to the same result: $\mathsf{Hash}(\mathsf{hk}, \mathrm{L}, W)$ with the hashing key and $\mathsf{ProjHash}(\mathsf{hp}, \mathrm{L}, W, w)$ with the projection key only but also a witness w that $W \in \mathrm{L}$. Of course, if $W \notin \mathrm{L}$, such a witness does not exist, and the smoothness property states that $\mathsf{Hash}(\mathsf{hk}, \mathrm{L}, W)$ is independent of hp. As a consequence, even knowing hp, one cannot guess $\mathsf{Hash}(\mathsf{hk}, \mathrm{L}, W)$.

Password-Authenticated Key Exchange (PAKE) protocols have received a lot of attention since the seminal paper of Bellovin and Merritt [BM92]. PAKE protocols indeed allow two players to agree on a common session key while using a simple password (a low-entropy secret) as authentication means. For such protocols, on-line dictionary attacks, for which testing a new password requires a new interaction, are unavoidable but their impact can be reduced by organizational means. They should be made the best possible attacks, and in particular, off-line dictionary attacks should definitely be prevented. Gennaro and Lindell [GL03] proposed a generic construction of PAKE in the Bellare-Pointcheval-Rogaway [BPR00] (BPR) security model. 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. Unfortunately, the language membership to check was then a Cramer-Shoup encryption of a specific word, and no SPHF was known for this language. They relaxed the initial Cramer-Shoup SPHFs (latter named CS-SPHFs) into GL-SPHFs that allow the projection key to be specific to the word whose language membership has to be proven.

* CNRS – UMR 8548 and INRIA – EPI Cascade

Their framework has thereafter been applied to the *Universal Composability* (UC) framework by Canetti, Halevi, Katz, Lindell and MacKenzie [CHK+05], and improved by Abdalla, Chevalier and Pointcheval [ACP09] to resist adaptive corruptions, still with GL-SPHFs. Since the projection keys could be generated only after having seen the word (a ciphertext of the password), three successive flows were a minimum.

More recently, one-round PAKE protocols have been proposed by Katz and Vaikuntanathan [KV11], still using the Gennaro-Lindell framework, but such that the commitment of the password and the projection key can be sent simultaneously, in one flow, by each player. Because of the independence of flows, new constraints appear on the SPHF, and a new definition was required: KV-SPHF, where the projection key depends on the hashing key only, as for CS-SPHF, and the smoothness holds even if the word is chosen after having seen the projection key. Katz and Vaikuntanathan first proposed a PAKE construction in the BPR model. Their construction received a much more efficient instantiation in [BBC+13], with only 6 group elements to be sent by each player.

Katz and Vaikuntanathan [KV11] also proposed a second construction provably secure against static corruptions in the UC framework. To this aim, each player additionally encrypts his hashing key to allow the key recovery by the simulator, so that it can compute the hash value even when a wrong password has initially been committed, whereas a success is expected. While this is the first one-round PAKE provably secure in the UC framework, hashing key recovery requires quite costly simulation-sound extractable NIZK (non-interactive zero-knowledge proof). Although, the latter can be improved by a more recent work [JR12], the UC-secure one-round PAKE is still much more costly than the BPR-secure protocol.

Zero-Knowledge Proofs/Arguments are essential tools in many cryptographic protocols. They are used to convince a verifier that some statement or word x is in a given $\mathcal{NP}$-language $\mathcal{L}$, defined by a polynomial time relation $\mathcal{R}$: $\mathcal{L} = \{x \mid \exists (w, y),\ \mathcal{R}(x, (w, y)) = 1\}$. This means that a word x is valid if there exists a witness (w, y) such that $\mathcal{R}(x, (w, y)) = 1$. The witness is divided in two parts w and y. We want to prove we know some w, *i.e.*, that some w can be extracted from a run of the protocol, such that there exists y, such that (w, y) is a valid witness. We use the notation of [CKS11] and write this as:

$$\text{ƛ}w,\ \exists y,\ \mathcal{R}(x, (w, y)) = 1.$$

This formalism generalizes both extractable arguments of knowledge (when $y = \perp$) and non-extractable zero-knowledge arguments (when $w = \perp$).

More precisely, we are interested in (partially) extractable zero-knowledge proofs or arguments (E-ZK) and their variants. E-ZK have to be *complete*, *sound*, *(partially) extractable* and *zero-knowledge*. Completeness states that an honest verifier always accepts a proof made by an honest prover for a valid statement and using a valid witness. Soundness states that no adversary can make an honest verifier accept a proof of a false statement x, either statistically (for *proofs*) or computationally (for *arguments*). However, in this paper, we do not distinguish between proofs and arguments and always talk about arguments. Partial extractability states that there exists an extractor able to simulate a verifier and to output a valid partial witness w from any successful interaction with an adversary playing the role of a prover. Finally, the zero-knowledge property ensures that it is possible to simulate a prover for any true statement x even without access to a witness (w, y) for this statement x.

In many protocols, the extractor and/or the zero-knowledge simulator have to rewind the proving adversary or the verifier adversary, respectively. In this paper, we will focus on protocols for which the extractability and the zero-knowledge properties are achieved without rewinding the players (and also in a black-box way). This is useful when such proofs are used in concurrent settings or as building blocks in protocols to be proven in the UC framework [Can01]. Formal definitions can be found in Appendix B.1.

Our Results. In this paper, we introduce a novel extension of SPHFs, called Trapdoor SPHF, or TSPHF. In addition to showing that an SPHF with an additional encryption of the hashing key and a simulation-sound extractable NIZK, as used in the UC-secure PAKE of Katz-Vaikuntanathan, can be seen as an inefficient

TSPHF, we provide efficient instantiations of TSPHF. To this aim, we extend the generic framework proposed in [BBC+13] for SPHFs to TSPHFs on languages of ciphertexts.

To illustrate efficiency of the TSPHFs, we use them in the one-round PAKE framework from [KV11]: we obtain a scheme which consists of 11 group elements in each direction (actually, 6 group elements in $\mathbb{G}_1$ and 5 group elements in $\mathbb{G}_2$ in an asymmetric bilinear setting, using the Cramer-Shoup encryption). It is secure in the UC framework against static corruptions under the SXDH assumption with a CRS, and just twice as more costly than the best BPR-secure PAKE [BBC+13], which makes it the most efficient one-round UC-secure PAKE to date.

While SPHFs can provide only honest-verifier zero-knowledge arguments (E-ZK, where the zero-knowledge protocol only holds when then verifiers is honest) for more expressive language than the Groth-Sahai methodology [GS08], and which are also more efficient than Groth-Sahai NIZK and sometimes rely on weaker assumptions, they do not lead to E-ZK. On the other hand, TSPHFs do help to construct efficient two-flow E-ZK protocols, for restricted languages, but at cost (in transmission complexity) comparable or better (depending on the exact language) than Groth-Sahai NIZK for the same languages. In addition, it is possible to slightly change these protocols to have an additional property called true-simulation extractabilty. Furthermore the E-ZK protocols and the true-simulation extractable variant have a cost comparable or better than Ω-protocols [GMY06], which are extractable variants of Σ-protocols, which are themself classical constructions for honest-verifier zero-knowledge arguments. A concrete instantiation of these constructions is an efficient E-ZK to prove the correct encryption of a valid Waters signature. This E-ZK can find applications in optimistic fair exchange of digital signatures [ASW98].

Outline. In Section 2, we recall the definition of SPHFs. Then in Section 3, we introduce the definition of TSPHF, with two constructions in Section 4. The latter is based on the generic framework for SPHFs introduced in [BBC+13]. In Section 5 we first apply this new tool to PAKE in the UC-framework. Eventually, we deal with zero-knowledge proofs, first for honest verifiers in Section 6, where SPHFs are enough, and then, in Section 7, we show how TSPHFs provide extractable zero-knowledge proofs. We provide a concrete instantiation to prove the correct encryption of a valid Waters signature in Section 8.

2 Preliminaries

2.1 General Definition of SPHFs

Let us consider a language $\mathrm{L} \subseteq \mathcal{S}et$, where words C are in L if there exists some witness w proving so. This language L is not the same as $\mathcal{L}$, the language considered for zero-knowledge arguments, in the introduction. A smooth projective hash function (SPHF) system for the language L is defined by four algorithms:

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

We write Π the set of hash values, which is supposed to be of size at least $2^{\mathfrak{K}}$, with $\mathfrak{K}$ the security parameter. The *correctness* of the SPHF assures that if $C \in \mathrm{L}$ with w a witness of this membership, then the two ways to compute the hash values give the same result: $\mathsf{Hash}(\mathsf{hk}, \mathrm{L}, C) = \mathsf{ProjHash}(\mathsf{hp}, \mathrm{L}, C, w)$. On the other hand, the security is defined through the *smoothness*, which guarantees that, if $C \notin \mathrm{L}$, the hash value is *statistically* indistinguishable from a random element of Π, even knowing hp. According to [BBC+13], depending on the exact definition of smoothness, there are three types of SPHFs:

- KV-SPHF: hp does not depend on C —*word-independent key*— and the smoothness holds even if C is chosen after having seen hp —*adaptive smoothness*—;

– CS-SPHF: hp does not depend on C —*word-independent key*— but the smoothness holds only if C is chosen before having seen hp —*non-adaptive smoothness*—;
– GL-SPHF: hp can depend on C —*word-dependent key*— and so the smoothness is correctly defined only if C is chosen before having seen hp —*non-adaptive smoothness*—.

2.2 SPHFs on Languages of Ciphertexts

Languages of Ciphertexts. In the following, we focus on languages of ciphertexts (LofC), where the witnesses are the random coins used for encryption of the plaintexts and possibly other values. The languages $\text{LofC}_{\mathsf{full\text{-}aux}}$ will thus be defined by $\mathsf{full\text{-}aux} = (\mathsf{crs}, \mathsf{aux})$, where crs will correspond to some fixed public parameters and aux will correspond to parameters which can change. The first part crs will at least contain the global parameters, the encryption key ek, and possibly additional parameters on the relation the plaintexts should satisfy. Contrary to the definition in [BBC+13], HashKG and ProjKG can use aux, since nothing is required to be kept private in the definition of the language. We write $\mathsf{Enc}(\ell, \mathsf{ek}, M; r)$ the encryption of the plaintext M with the optional label ℓ and the random coins r.

Labeled Cramer-Shoup Encryption Scheme (CS). In this article, we use the ElGamal [ElG85] IND-CPA encryption scheme and the Cramer-Shoup [CS98] labeled IND-CCA encryption scheme.

Here, we briefly review the CS labeled encryption scheme, where we combine all the public information in the encryption key. We thus have a group $\mathbb{G}$ of prime order p, with two independent generators $(g_1, g_2) \xleftarrow{\$} \mathbb{G}^2$, a hash function $\mathfrak{H}_K \xleftarrow{\$} \mathcal{H}$ from a collision-resistant hash function family onto $\mathbb{Z}_p^*$, and a reversible mapping $\mathcal{G}$ from $\{0,1\}^n$ to $\mathbb{G}$. From 5 scalars $(x_1, x_2, y_1, y_2, z) \xleftarrow{\$} {\mathbb{Z}_p}^5$, one also sets $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$. The encryption key is $\mathsf{ek} = (\mathbb{G}, g_1, g_2, c, d, h, \mathfrak{H}_K)$, while the decryption key is $\mathsf{dk} = (x_1, x_2, y_1, y_2, z)$. For a message $m \in \{0,1\}^n$, with $M = \mathcal{G}(m) \in \mathbb{G}$, the labeled Cramer-Shoup ciphertext is:

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

with $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e) \in \mathbb{Z}_p^*$.

Vector Encryptions. As explained in Appendix A.3, to encrypt vectors of messages, as we will do in the sequel, one can concatenate independent ElGamal or Cramer-Shoup ciphertexts (with some common parameter ξ for the Cramer-Shoup encryption scheme in order to keep the global non-malleability) or re-use random coins [BBS03]. This second method yields more compact ciphertexts at the expense of slightly larger keys.

3 Definition of Computational Smoothness and TSPHF

In this section, we introduce TSPHFs, which are SPHFs with a trapdoor enabling a simulator to compute the hash value of any valid word C without knowing hk nor any witness, but only knowing hp. TSPHFs also provide a way to ensure that hp is valid, which prevents the attack against the witness-indistinguishability described in the previous section. It can be seen that, intuitively, in most cases, a TSPHF cannot be statistically smooth, and so, before introducing TSPHFs, we need to introduce a new notion of smoothness: computational smoothness.

3.1 Computationally-Smooth SPHF

Let us first suppose there exists an algorithm Setup which takes as input the security parameter $\mathfrak{K}$ and outputs a CRS crs together with a trapdoor τ, which is not the trapdoor of the TSPHF, but just a trapdoor of crs. The

Initialize(1^λ)
  $(\mathsf{crs}, \tau) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$
  return crs, τ

ProjKG(aux, C)
  $(\mathsf{aux}', C') \leftarrow (\mathsf{aux}, C)$
  $\mathsf{full\text{-}aux} \leftarrow (\mathsf{crs}, \mathsf{aux})$
  $\mathsf{hk} \xleftarrow{\$} \mathsf{HashKG}(\mathsf{full\text{-}aux})$
  $\mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{full\text{-}aux}, C)$
  return hp

Hash(C)
  $\mathsf{aux} \leftarrow \mathsf{aux}'$; $\mathsf{full\text{-}aux} \leftarrow (\mathsf{crs}, \mathsf{aux})$
  $C \leftarrow C'$ ▷ if non-adaptively-smooth SPHF
  if $b = 0$ or $C \in \text{LoFC}_{\mathsf{full\text{-}aux}}$ **then**
    $H \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C)$
  else $H \xleftarrow{\$} \Pi$
  return H

Finalize(b')
  return b'

Fig. 1. Games $\mathsf{Exp}^{\text{smooth}-b}_{\mathfrak{K}}(\mathcal{A})$ ($b = 0$ or 1) for computational smoothness

trapdoor τ can be $\perp$, but in our article, the trapdoor will contain at least the decryption key of the encryption scheme, and possibly other data such that, for any $C \in \mathcal{S}et$, it is possible to check whether $C \in \text{LoFC}_{\mathsf{full\text{-}aux}}$ or not, in polynomial time.

Let us then consider the two games $\mathsf{Exp}^{\text{smooth}-b}_{\mathfrak{K}}(\mathcal{A})$ (with $b = 0$ or 1) depicted in Figure 1, where Π denotes the set of hash values. There are two variants of the games: whether the SPHF is adaptively-smooth (KV-SPHF) or not (CS-SPHF and GL-SPHF).

Let us first explain the games for a non-adaptively-smooth SPHF. The procedure **Initialize** generates and outputs the CRS crs and its trapdoor τ. It is important to notice that computational smoothness has to hold even when the adversary knows the trapdoor, and so may depend on what is in the trapdoor τ.

During the execution of the game, the adversary is allowed to make one query **ProjKG**(aux, C) to get a projection key hp associated with aux and C, and then one query **Hash**$(\perp)$ to get the hash value of C. If $C \in \text{LoFC}_{\mathsf{full\text{-}aux}}$, smoothness does not apply, thus **Hash**(C) really returns the hash value H of C: $H = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C)$, for a hashing key hk associated with hp. Otherwise, the smoothness should apply with a real-or-random indistinguishability game, and thus, if $b = 0$ the real hash value is returned too, whereas a random value in Π is returned when $b = 1$. Eventually, the adversary ends the game by querying the **Finalize** procedure with its guess b' for b. We remark that the procedure **Hash** may or may not be polynomial time, depending on τ, since it is not necessarily possible to efficiently check whether $C \in \text{LoFC}_{\mathsf{full\text{-}aux}}$.

For the adaptively-smooth variant, the adversary does not need to provide the word C when it makes a query to **ProjKG**. It gives $\perp$ instead and can choose C adaptively after having seen hp, as input to the **Hash** query.

Formally, an SPHF is (t, ε)-smooth if for all adversary $\mathcal{A}$ running in time at most t:

$$\left|\Pr\left[\mathsf{Exp}^{\text{smooth}-1}_{\mathfrak{K}}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{smooth}-0}_{\mathfrak{K}}(\mathcal{A}) = 1\right]\right| \leq \varepsilon.$$

The classical statistical-smoothness implies the (t, ε)-smoothness for any t, and any non-negligible ε (and whatever is the trapdoor τ).

3.2 Trapdoor SPHF

A TSPHF 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 τ' specific to crs′, which can be used to compute the hash value of words C knowing only hp. For TSPHF, we assume $\mathsf{full\text{-}aux} = (\mathsf{crs}, \mathsf{crs}', \mathsf{aux})$, although the language $\text{LoFC}_{\mathsf{full\text{-}aux}}$ still does not depend on crs′. Formally, a TSPHF is defined by seven algorithms:

– TSetup(crs) takes as input the CRS crs (generated by Setup) and generates the second CRS crs′, together with a trapdoor τ';

- HashKG, ProjKG, Hash, and ProjHash behave as for a classical SPHF;
- VerHP(hp, full-aux, C) outputs 1 if hp is a valid projection key, and 0 otherwise. When hp does not depend on C (word-independent key), the input C can be replaced by $\perp$;
- THash(hp, full-aux, C, τ') outputs the hash value of C from the projection key hp and the trapdoor τ'.

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 $C \in \mathcal{S}et$, if hk and hp are honestly generated, we have: VerHP(hp, full-aux, C) = 1 and Hash(hk, full-aux, C) = THash(hp, full-aux, C, τ'), with overwhelming probability;
- *Smoothness* is exactly the same as for SPHFs, except that in the **Initialize** procedure, TSetup is also called, but while τ' is dropped, crs$'$ is forwarded to the adversary (together with crs and τ);
- The (t, ε)-*soundness* property says that, given crs, τ and crs$'$, no adversary running in time at most t can produce a projection key hp, a value aux, a word C and valid witness w such that hp is valid (*i.e.*, VerHP(hp, full-aux, C) = 1) but THash(hp, full-aux, C, τ') $\neq$ ProjHash(hp, full-aux, 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 τ is not an input of THash and it is possible to use THash, while generating crs with an algorithm which cannot output τ (as soon as the distribution of crs output by this algorithm is indistinguishable from the one output by Setup, obviously). For example, if τ contains a decryption key, it is still possible to use the IND-CPA game for the encryption scheme, while making calls to THash.

4 Constructions of TSPHFs

In this section, we show two ways of constructing TSPHFs from SPHFs. The first way uses NIZK, and works for any SPHF but is very inefficient for generic SPHFs, and inefficient for most SPHFs. This is essentially the approach of [KV11]. The second way is much more efficient and works for most SPHFs based on the generic framework for SPHFs introduced in [BBC+13] and recalled in Section E.1.

4.1 Construction of TSPHFs using NIZK

A naive solution to transform any SPHF into a TSPHF consists in replacing the projection key hp by a pair (hp, π), where π is an ENIZK (a non-interactive E-ZK) proof of the knowledge ($\mathcal{K}$) of a hashing key hk such that hp is the projection key of hk. In Appendix F.1, we show that this provides a correct, smooth and sound TSPHF. Intuitively the hash correctness directly comes from the correctness of the original SPHF, the trapdoor correctness and the soundness come from the extractability of the ENIZK (and may not be perfect) and the smoothness comes from the zero-knowledge property of the ENIZK.

In Appendix F.1, we also show some improvements for this naive construction to make quite efficient TSPHF, and in particular to avoid having to do bit-by-bit Groth-Sahai ENIZK. These improvements can be seen as a generalization of the method proposed by Jutla and Roy in [JR12, Section 8]. But even with these improvements, this naive construction is still less efficient than the constructions described in the sequel.

4.2 Generic Framework for GL-SPHF/KV-SPHF

In [BBC+13], the authors introduced a formal framework for SPHFs using a new notion of graded rings, derived from [GGH12]. It enables to deal with cyclic groups, bilinear groups (with symmetric or asymmetric pairings), or even groups with multi-linear maps. In particular, it helps to construct concrete SPHFs for quadratic pairing equations over ciphertexts, which enable to construct efficient LAKE [BBC+13] for any language handled by the Groth-Sahai NIZKs, and so for any $\mathcal{NP}$-language.

This generic framework is recalled in Appendix E.1. For the sake of simplicity, we focus here on cyclic groups, with the basic intuition only, and provide some illustrations. While we keep the usual multiplicative notation for the cyclic group $\mathbb{G}$, we use an extended notation: $r \odot u = u \odot r = u^r$, for $r \in \mathbb{Z}_p$ and $u \in \mathbb{G}$, and $u \oplus v = u \cdot v$, for $u, v \in \mathbb{G}$. Basically, $\oplus$ and $\odot$ correspond to the addition and the multiplication in the exponents, that are thus both commutative. We then extend this notation in a natural way when working on vectors and matrices.

Our goal is to deal with languages of ciphertexts $\textsc{Lofc}_{\mathsf{full\text{-}aux}}$: we assume that crs is fixed and we write $\mathrm{L}_{\mathsf{aux}} = \textsc{Lofc}_{\mathsf{full\text{-}aux}} \subseteq \mathcal{S}et$ where $\mathsf{full\text{-}aux} = (\mathsf{crs}, \mathsf{aux})$.

Language Representation. For a language $\mathrm{L}_{\mathsf{aux}}$, we assume there exist two positive integers k and n, a function $\Gamma : \mathcal{S}et \mapsto \mathbb{G}^{k \times n}$, and a family of functions $\Theta_{\mathsf{aux}} : \mathcal{S}et \mapsto \mathbb{G}^{1 \times n}$, such that for any word $C \in \mathcal{S}et$, $(C \in \mathrm{L}_{\mathsf{aux}}) \iff (\exists \boldsymbol{\lambda} \in \mathbb{Z}_p^{1 \times k}$ such that $\Theta_{\mathsf{aux}}(C) = \boldsymbol{\lambda} \odot \Gamma(C))$. In other words, we assume that $C \in \mathrm{L}_{\mathsf{aux}}$, if and only if, $\Theta_{\mathsf{aux}}(C)$ is a linear combination of (the exponents in) the rows of some matrix $\Gamma(C)$. For a $\mathsf{KV\text{-}SPHF}$, Γ is supposed to be a constant function (independent of the word C). Otherwise, one gets a $\mathsf{GL\text{-}SPHF}$.

We furthermore require that a user, who knows a witness w of the membership $C \in \mathrm{L}_{\mathsf{aux}}$, can efficiently compute the above *linear* combination $\boldsymbol{\lambda}$. This may seem a quite strong requirement but this is actually verified by very expressive languages over ciphertexts such as ElGamal, Cramer-Shoup and variants.

We briefly illustrate it on a $\mathsf{KV\text{-}SPHF}$ for the language of CS ciphertexts encrypting a message $M = \mathsf{aux}$. Words in the language $\mathrm{L}_{\mathsf{aux}}$ are ciphertexts $C = (u_1 = g_1^r, u_2 = g_2^r, e = M \cdot h^r, v = (cd^\xi)^r)$, with $r \in \mathbb{Z}_p$ and $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e) \in \mathbb{Z}_p^*$. We choose $k = 2$, $\mathsf{aux} = M$, $n = 5$, and:

$$\Gamma = \begin{pmatrix} g_1 & 1 & g_2 & h & c \\ 1 & g_1 & 1 & 1 & d \end{pmatrix} \qquad \boldsymbol{\lambda} = (r, r\xi) \qquad \begin{array}{r} \boldsymbol{\lambda} \odot \Gamma = (g_1^r, g_1^{r\xi}, g_2^r, h^r, (cd^\xi)^r) \\ \Theta_M(C) = (u_1, u_1^\xi, u_2, e/M, v). \end{array}$$

Essentially, one tries to make the first columns of $\Gamma(C)$ and the first components of $\Theta_{\mathsf{aux}}(C)$ to completely determine $\boldsymbol{\lambda}$. In our illustration, the first two columns with $u_1 = g_1^r$ and $u_1^\xi = g_1^{r\xi}$ really imply $\boldsymbol{\lambda} = (r, r\xi)$, and the three last columns help to check the language membership: we want $u_2 = g_2^r$, $e/M = h^r$, and $v = (cd^\xi)^r$, with the same r as for u_1.

Smooth Projective Hash Function. With the above notations, the hashing key is a vector $\mathsf{hk} = \boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^\intercal \xleftarrow{\$} \mathbb{Z}_p^n$, while the projection key is, for a word C, $\mathsf{hp} = \boldsymbol{\gamma}(C) = \Gamma(C) \odot \boldsymbol{\alpha} \in \mathbb{G}^k$ (if Γ depends on C, this leads to a $\mathsf{GL\text{-}SPHF}$, otherwise, one gets a $\mathsf{KV\text{-}SPHF}$). Then, the hash value is:

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

Our above Γ, $\boldsymbol{\lambda}$, and Θ_M immediately lead to the $\mathsf{KV\text{-}SPHF}$ on CS, introduced in [BBC+13]: with $\mathsf{hk} = (\eta_1, \eta_2, \theta, \mu, \nu) \xleftarrow{\$} \mathbb{Z}_p^5$, the product with Γ leads to: $\mathsf{hp} = (\mathsf{hp}_1 = g_1^{\eta_1} g_2^\theta h^\mu c^\nu, \mathsf{hp}_2 = g_1^{\eta_2} d^\nu) \in \mathbb{G}^2$, and

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

A security analysis that proves the above generic SPHF is perfectly smooth can be found in [BBC+13]. Intuitively, for a word $C \notin \mathrm{L}_{\mathsf{aux}}$ and a projection key $\mathsf{hp} = \boldsymbol{\gamma}(C) = \Gamma(C) \odot \boldsymbol{\alpha}$, the vector $\Theta_{\mathsf{aux}}(C)$ is not in the *linear* span of $\Gamma(C)$, and thus $H = \Theta_{\mathsf{aux}}(C) \odot \boldsymbol{\alpha}$ is independent from $\Gamma(C) \odot \boldsymbol{\alpha} = \mathsf{hp}$.

4.3 Efficient Construction of TSPHFs under DDH

We now explain how to construct a TSPHF in a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, from any SPHF constructed via the above framework, provided the SPHF does not require pairings (as all the SPHFs described in this

paper), and under an additional assumption detailed later (for the smoothness to hold). To this aim, we extend our notations with $g_1 \odot g_2 = g_2 \odot g_1 = e(g_1, g_2)$, and scalars can operate on any group element as before. Intuitively, our TSPHF construction is such that all the "SPHF" part of the TSPHF is in $\mathbb{G}_1$, whereas the trapdoor part is in $\mathbb{G}_2$. And the trapdoor part simply contains some representation of $\boldsymbol{\alpha}$, representation which cannot be used without knowing the trapdoor τ'.

The second CRS is a random element $\mathsf{crs}' = \zeta \xleftarrow{\$} \mathbb{G}_2$, and its trapdoor is its discrete logarithm τ', such that $\zeta = g_2^{\tau'} = \tau' \odot g_2$. The hashing key $\mathsf{hk} = \boldsymbol{\alpha}$ is the same as before. The projection key is the ordered pair $\mathsf{hp} = (\boldsymbol{\gamma}, \boldsymbol{\chi})$, where $\boldsymbol{\gamma}$ is the same as before, and $\boldsymbol{\chi} = \zeta \odot \boldsymbol{\alpha}$. The projection key is valid (*i.e.*, $\mathsf{VerHP}(\mathsf{hp}, \mathsf{full\text{-}aux}, C) = 1$) if and only if

$$\boldsymbol{\chi} \in \mathbb{G}_2^n \qquad \text{and} \qquad \zeta \odot \boldsymbol{\gamma} = \Gamma \odot \boldsymbol{\chi}, \tag{1}$$

Then, for any word $C \in \mathrm{L}_{\mathsf{full\text{-}aux}}$ with witness w corresponding to the vector $\boldsymbol{\lambda}$, the hash value is

$$\mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C) \stackrel{\text{def}}{=} \Theta(C) \odot \boldsymbol{\alpha} \odot g_2 = \boldsymbol{\lambda} \odot \boldsymbol{\gamma} \odot g_2 \stackrel{\text{def}}{=} \mathsf{ProjHash}(\mathsf{hp}, \mathsf{full\text{-}aux}, C, w).$$

Equation (1) means that $\boldsymbol{\chi}$ can be written $\boldsymbol{\chi} = \tau' \odot \boldsymbol{\alpha'}$, with $\boldsymbol{\alpha'} \in \mathbb{Z}_p^n$ verifying $\boldsymbol{\gamma} = \Gamma \odot \boldsymbol{\alpha'}$, *i.e.*, $\mathsf{hk}' = \boldsymbol{\alpha'}$ is a valid hashing key for $\boldsymbol{\gamma}$. We do not have necessarily $\boldsymbol{\alpha} = \boldsymbol{\alpha'}$, however, for any word $C \in \mathrm{L}_{\mathsf{full\text{-}aux}}$, we have and we set

$$\boldsymbol{\lambda} \odot \boldsymbol{\gamma} \odot g_2 = \Theta(C) \odot \boldsymbol{\alpha'} \odot g_2 = \tau'^{-1} \odot \Theta(C) \odot \boldsymbol{\chi} \stackrel{\text{def}}{=} \mathsf{THash}(\mathsf{hp}, \mathsf{full\text{-}aux}, C, \tau').$$

In Appendix E.3, we prove the resulting TSPHF is computationally smooth under the DDH assumption in $\mathbb{G}_2$, if the discrete logarithms of $\Gamma_{\mathsf{aux}}(C)$ can be computed from τ. This latter assumption on $\Gamma_{\mathsf{aux}}(C)$ and τ is required for technical reasons in the proof of smoothness. The correctness and the perfect soundness are easy to prove from the construction, and so the resulting TSPHF is correct, smooth and sound.

4.4 TSPHF on Cramer-Shoup Ciphertexts

To illustrate this generic transformation, we apply it to extend the KV-SPHF on Cramer-Shoup ciphertexts (the example in Section 4.2) into a TSPHF. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be a bilinear group. We consider the same language and use the same notations as in Section 4.2 except we replace $\mathbb{G}$ by $\mathbb{G}_1$, g_1 and g_2 by $g_{1,1}$ and $g_{1,2}$ resp., and h by h_1, while g_2 is a generator of $\mathbb{G}_2$.

To get a TSPHF, we choose a random scalar τ' in $\mathbb{Z}_p$ and set $\mathsf{crs}' = \zeta = g_2^{\tau'}$. Then the hashing key, the projection key and the hash value of the TSPHF are defined as follows:

$$\begin{aligned}
&\mathsf{hk} = (\eta_1, \eta_2, \theta, \mu, \nu) \xleftarrow{\$} \mathbb{Z}_p^5\\
&\mathsf{hp} = (\mathsf{hp}_1 = g_{1,1}^{\eta_1} g_{1,2}^{\theta} h_1^{\mu} c^{\nu}, \mathsf{hp}_2 = g_{1,1}^{\eta_2} d^{\nu}, \mathsf{hp}_3) \in \mathbb{G}_1^2 \times \mathbb{G}_2^5\\
&\quad \text{where } \mathsf{hp}_3 = (\chi_{1,1} = \zeta^{\eta_1}, \chi_{1,2} = \zeta^{\eta_2}, \chi_2 = \zeta^{\theta}, \chi_3 = \zeta^{\mu}, \chi_4 = \zeta^{\nu}) \in \mathbb{G}_2^5
\end{aligned}$$

$$\begin{aligned}
\mathsf{Hash}(\mathsf{hk}, (\mathsf{ek}, m), C) &= e(u_1{}^{(\eta_1 + \xi\eta_2)} u_2{}^{\theta} (e/\mathcal{G}(m))^{\mu} v^{\nu}, g_2)\\
\mathsf{ProjHash}(\mathsf{hp}, (\mathsf{ek}, m), C, r) &= e((\mathsf{hp}_1 \mathsf{hp}_2{}^{\xi})^r, g_2)
\end{aligned}$$

The projection key is valid if and only if: $e(\mathsf{hp}_1, \zeta) = e(g_{1,1}, \chi_{1,1}) \cdot e(g_{1,2}, \chi_2) \cdot e(h_1, \chi_3) \cdot e(c, \chi_4)$ and $e(\mathsf{hp}_2, \zeta) = e(g_{1,1}, \chi_{1,2}) \cdot e(d, \chi_4)$. For any $C \in \mathrm{LoFC}_{(\mathsf{crs},\mathsf{aux})}$, the hash value can be computed from C and τ' as

$$\mathsf{THash}(\mathsf{hp}, (\mathsf{ek}, m), C, \tau') = \Big(e(u_1, \chi_{1,1} \cdot \chi_{1,2}^{\xi}) \cdot e(u_2, \chi_2) \cdot e(e/\mathcal{G}(m), \chi_3) \cdot e(v, \chi_4)\Big)^{1/\tau'}.$$

The resulting TSPHF is smooth under the DDH in $\mathbb{G}_2$, hence the global SXDH assumption.

5 One-Round UC-Secure PAKE from TSPHFs

In this section, we introduce a direct application of the previous KV-TSPHF on Cramer-Shoup ciphertexts: an efficient UC-secure PAKE. Before that, we describe a generic UC-secure one-round PAKE scheme from any IND-CCA encryption scheme with an associated TSPHF. Our efficient UC-secure PAKE is an instantiation of this generic PAKE.

5.1 Generic One-Round UC-Secure PAKE

The ideal functionality of a Password-Authenticated Key Exchange (PAKE) is depicted in Figure 2. It has been proposed in [CHK+05].

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

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

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

Fig. 2. Ideal Functionality for PAKE $\mathcal{F}_{\text{pwKE}}$

Our generic PAKE is a slight variant of the one-round PAKE from [KV11], where the SPHF and the SS-NIZK are replaced by a TSPHF. It is depicted in Figure 3 It is secure in the UC framework against static corruptions, with a common reference string for any TSPHF on the language of a valid ciphertext on a message m under an IND-CCA-secure labeled encryption scheme. The full proof is provided in Appendix D.1. It is in the same vein as the KV's proof but a bit more intricate for two reasons: we do not assume a prior agreement of the session ID which makes our scheme a truly one-round protocol; our TSPHF does not guarantee the smoothness (even computationally) when the trapdoor τ' is known, and then, we have to modify the order of the games to use this trapdoor at the very end only.

One can remark that the original scheme in [KV11] can be seen as an instantiation of our scheme with a naive TSPHF based on NIZK (Section 4.1). Therefore, the security of the original KV's PAKE protocol is actually implied by our proof. And our proof also shows that their construction can be simplified by removing the commitment of hk and replacing the SS-NIZK by an ENIZK for the knowledge of hk. However, even with these improvements and the improvements of Appendix F.1, the resulting construction is still less efficient than the one just below.

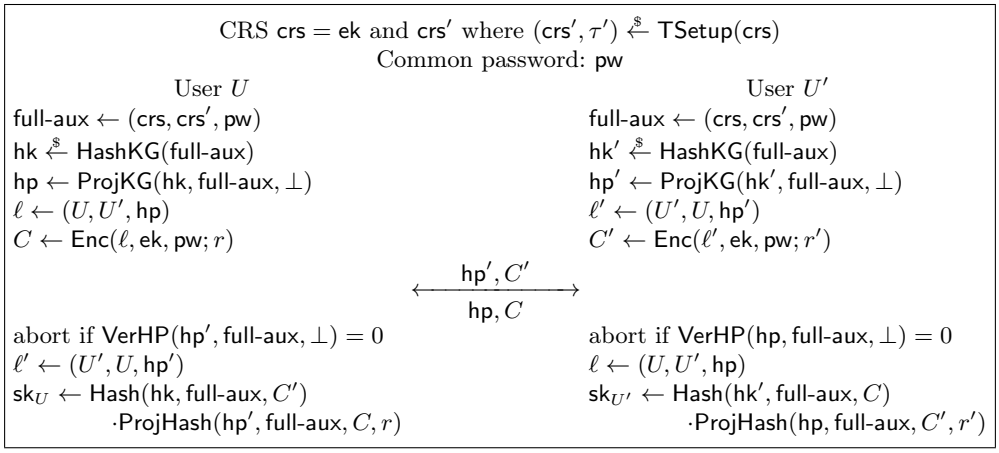

Fig. 3. Generic UC-Secure One-Round PAKE

5.2 Efficient Instantiation

Let us now instantiate this generic PAKE with a Cramer-Shoup encryption scheme and our KV-TSPHF on Cramer-Shoup ciphertexts. The resulting scheme is depicted in Figure 4. The communication complexity is of 6 elements in $\mathbb{G}_1$ and 5 elements in $\mathbb{G}_2$ only in each direction, which makes it the most efficient one-round UC-secure PAKE to date.

6 Zero-Knowledge Arguments from SPHFs

In this section, we show how to construct honest-verifier zero-knowledge arguments (HVE-ZK) from SPHFs. A HVE-ZK is a weak variant of E-ZK, for which the zero-knowledge property only needs to hold with honest verifiers.

We then present the limitations of this construction, namely the fact that the resulting argument is even not witness-indistinguishable, *i.e.*, a malicious prover may be able to distinguish which witness has been used by an honest prover[1], in general.

6.1 Honest-Verifier Zero-Knowledge Arguments from SPHFs

The idea of the construction is that a prover, who knows some valid statement x together with a valid witness (w, y), encrypts w, using an IND-CPA encryption scheme, in some ciphertext C, under some encryption key ek contained in crs. Then, using an SPHF, he shows that the ciphertext C is an encryption of a valid partial witness w for the word x: the verifier chooses some hashing key hk and sends the corresponding projection key hp to the prover; the prover sends back the hash value H of the ciphertext C computed from hp, w, y and the random coins used in C, using ProjHash; and the verifier checks he gets the same hash value from hk, using Hash. If the SPHF is a KV-SPHF, the prover can send the ciphertext C together with H after receiving hp from the verifier. This yields a two-flow protocol. More precisely, we use a KV-SPHF for the following language:

$$\textsc{Lofc}_{\mathsf{full\text{-}aux}} = \{C \mid \exists w,\, \exists r,\, \exists y,\; C = \mathsf{Enc}(\mathsf{ek}, w; r) \text{ and } \mathcal{R}(x, (w, y))\},$$

where aux is the statement x, and crs contains the encryption key ek and possibly some global parameters related to the language $\mathcal{L}$ associated with the relation $\mathcal{R}$. The complete protocol is depicted in Figure 5.

[1] The formal definition of witness-indistinguishability can be found in Appendix B.1.

CRS: $\mathsf{ek} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_{1,1}, g_{1,2}, c, d, h, \mathfrak{H}_K), \zeta \in \mathbb{G}_2$ Common password: pw

Both users do the same:

- U, with expected partner U', generates hk, hp and C:

$$\mathsf{hk} = (\alpha_{u_1,1}, \alpha_{u_1,2}, \alpha_{u_2}, \alpha_e, \alpha_v) \xleftarrow{\$} \mathbb{Z}_p^5$$
$$\mathsf{hp} = (\mathsf{hp}_1 = g_{1,1}^{\alpha_{u_1,1}} g_{1,2}^{\alpha_{u_2}} h^{\alpha_e} c^{\alpha_v}, \mathsf{hp}_2 = g_{1,1}^{\alpha_{u_1,2}} d^{\alpha_v}, \mathsf{hp}_3) \in \mathbb{G}_1^2 \times \mathbb{G}_2^5$$
$$\text{where } \mathsf{hp}_3 = (\chi_{u_1,1} = \zeta^{\alpha_{u_1,1}}, \chi_{u_1,2} = \zeta^{\alpha_{u_1,2}}, \chi_{u_2} = \zeta^{\alpha_{u_2}}, \chi_e = \zeta^{\alpha_e}, \chi_v = \zeta^{\alpha_v}) \in \mathbb{G}_2^5$$
$$C = (\ell, \mathbf{u} = (g_{1,1}^r, g_{1,2}^r), e = \mathcal{G}(\mathsf{pw}) \cdot h^r, v = (cd^\xi)^r) \in \mathbb{G}_1^4 \qquad \text{where } \ell = (U, U', \mathsf{hp}) \text{ and } \xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$$

- U sends $\mathsf{hp} \in \mathbb{G}_1^2 \times \mathbb{G}_2^5$ and $C \in \mathbb{G}_1^4$
- Upon receiving $\mathsf{hp}' = (\mathsf{hp}'_1, \mathsf{hp}'_2, \chi'_{u_1,1}, \chi'_{u_1,2}, \chi'_{u_2}, \chi'_e, \chi'_v) \in \mathbb{G}_1^2 \times \mathbb{G}_2^5$ and $C' = (\mathbf{u}' = (u'_1, u'_2), e', v') \in \mathbb{G}_1^4$, U checks whether

$$e(\mathsf{hp}'_1, \zeta) = e(g_{1,1}, \chi'_{u_1,1}) \cdot e(g_{1,2}, \chi'_{u_2}) \cdot e(h, \chi'_e) \cdot e(c, \chi'_v) \qquad \text{and} \qquad e(\mathsf{hp}'_2, \zeta) = e(g_{1,1}, \chi'_{u_1,2}) \cdot e(d, \chi'_v)$$

- If one equality does not hold, U aborts,otherwise U sets $\ell' = (U', U, \mathsf{hp}')$ and $\xi' = \mathfrak{H}_K(\ell', \mathbf{u}', e')$
- U computes

$$\mathsf{sk}_U = e(u_1'^{(\alpha_{u_1,1} + \xi' \alpha_{u_1,2})} u_2'^{\alpha_{u_2}} (e'/\mathcal{G}(\mathsf{pw}))^{\alpha_e} v'^{\alpha_v}, g_2) \cdot e((\mathsf{hp}'_1 \mathsf{hp}'^{\xi}_2)^r, g_2)$$

Fig. 4. UC-Secure One-Round PAKE based on DDH

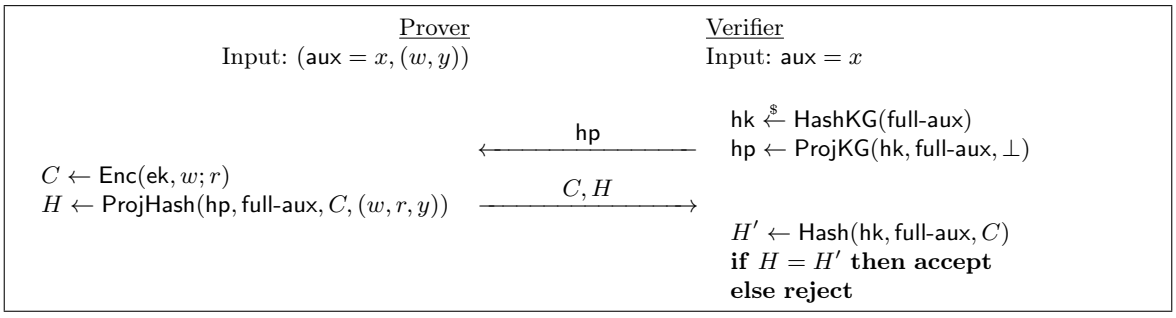

Fig. 5. HVE-ZK Argument from KV-SPHFs

It is possible to use a GL-SPHF instead of a KV-SPHF for the above language, if the ciphertext C is sent before hp. The protocol becomes three-flow but can require fewer bits to be transmitted, because GL-SPHFs are often more efficient than KV-SPHFs. It is depicted in Figure 6.

Completeness comes from the correctness of the SPHF and soundness comes from the statistical smoothness of the SPHF. The extractor just acts as an honest verifier and decrypts the ciphertext C of the adversarial prover at the end. The simulator for the honest-verifier zero-knowledge property just encrypts an arbitrary value in C and computes H using hk: $H = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C)$. The IND-CPA property of the encryption scheme used for C ensures the simulator transcripts are computationally indistinguishable from real transcripts, and so the proposed construction is honest-verifier zero-knowledge.

6.2 Instantiations

Multi-Exponentiation Equations in Cyclic Group $\mathbb{G}$. In Appendix C.1, we show a KV-SPHF which yields an efficient HVE-ZK for the following language:

$$\mathcal{H}(X_1, \ldots, X_n) \in \mathbb{G}^n, \exists (y_1, \ldots, y_m) \in \mathbb{Z}_p^m, \forall k \in \{1, \ldots, t\}, \prod_{i=1}^{n} X_i^{a_{k,i}} = \prod_{j=1}^{m} A_{k,j}^{y_j} \cdot B_k,$$

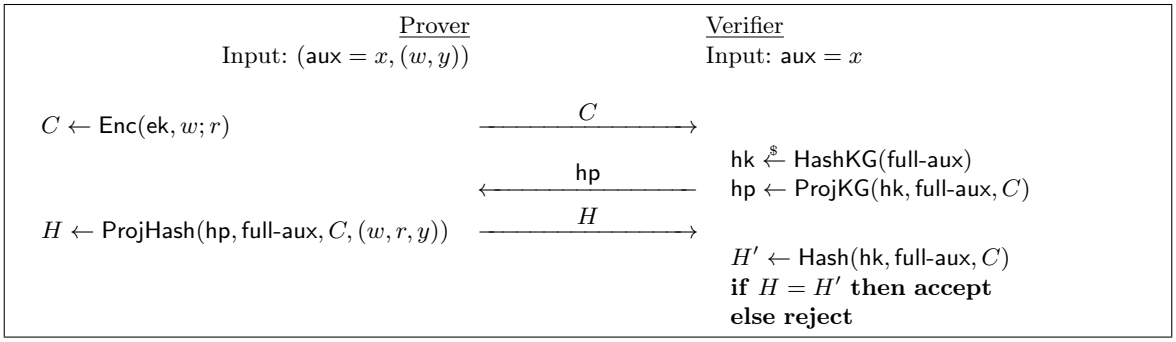

Fig. 6. HVE-ZK Argument from GL-SPHFs

where a statement x is a tuple containing all the constants $a_{k,i}$, $A_{k,j}$ and B_k, or some of them (in this case, the other constants are in crs).

Let us now compare the transmission complexity of the resulting HVE-ZK, when ElGamal ciphertexts with reuse of randomness are used, with the transmission complexity of Groth-Sahai NIZK [GS08] and Ω-protocols [GMY06]. The ciphertext C of $(X_1, \ldots, X_n)$ requires $n+1$ elements in $\mathbb{G}$, the projection key hp requires $m+1$ elements in $\mathbb{G}$ and the hash value H is 1 element in $\mathbb{G}$, which gives a total of $n+m+3$ elements in $\mathbb{G}$.

For the same language, the corresponding Groth-Sahai NIZK cannot make reuse of randomness, requires to use bilinear groups $(p, \mathbb{G} = \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ and to commit $(g_2^{y_j})_j$, and so requires at least n more elements in $\mathbb{G}_1$, $2m$ more elements in $\mathbb{G}_2$ (and m fewer in $\mathbb{G}_1$ but elements of $\mathbb{G}_2$ are at least twice larger than elements in $\mathbb{G}_1$), but is non-interactive and zero-knowledge instead of two-flow and honest-verifier zero-knowledge. We also remark that Groth-Sahai NIZK requires the SXDH assumption, whereas our construction only needs the DDH assumption in $\mathbb{G}$. The corresponding Ω-protocol, which is an HVE-ZK as our protocol, uses $n+t+2$ elements in $\mathbb{G}$ and $m+2$ in $\mathbb{Z}_p$, which is $t+1$ elements more than our protocol, as shown in Appendix C.2.

A detailed and concrete application of this HVE-ZK for multi-exponentiation equations, will be given in Section 8, together with a concrete comparison with Ω-protocols and Groth-Sahai NIZK.

Pairing Product Equations in Bilinear Groups. SPHF constructions in [BBC+13] yield efficient two-flow HVE-ZK able to deal with systems of pairing product equations. These systems are more expressive than the languages by Groth-Sahai NIZK in [GS08]: in addition to all what can be done using these NIZK, they also handle unknowns in $\mathbb{G}_T$.

6.3 Limitations of SPHFs

Unfortunately, without any extra property on the SPHF, the above construction is not witness indistinguishable, and so not zero-knowledge, in general. The main problem is that, for some SPHFs, it may be possible to generate hp in such a way that the hash value H computed by the prover (through ProjHash) depends on the witness used. This happens, in particular, when the language $\text{LoC}_{\mathsf{full\text{-}aux}}$ of the SPHF (and also the language $\mathcal{L}$ of the HVE-ZK) is a disjunction of two languages and when the generic construction of [ACP09] for disjunctions is used to construct the SPHF.

Let us indeed consider the following language: νX, $X = X_0$ or $X = X_1$, with X_0 and X_1 two distinct constants, which are two distinct witnesses. This language is completely trivial, but our attack works for more interesting cases such as languages νX such that X is a signature on a message m_0 or on another message m_1.

For the previous HVE-ZK construction, we need an SPHF for the following language:

$$\text{LoC}_{\mathsf{full\text{-}aux}} = \{C \mid \exists r,\, C = \mathsf{Enc}(\mathsf{ek}, X_0; r) \text{ or } C = \mathsf{Enc}(\mathsf{ek}, X_1; r)\}.$$

Suppose we have two SPHFs $(\mathsf{HashKG}_b, \mathsf{ProjKG}_b, \mathsf{Hash}_b, \mathsf{ProjHash}_b)$ for the languages of ciphertexts of X_b, for $b \in \{0,1\}$ and suppose we use the generic disjunction method of [ACP09] to construct an SPHF for $\text{LOFC}_{\mathsf{full\text{-}aux}}$. The resulting SPHF works as follows. Let hk_0 and hk_1 be two hashing keys of the previous SPHFs, and hp_0 and hp_1 the corresponding projection keys (generated by $\mathsf{HashKG}_0, \mathsf{HashKG}_1, \mathsf{ProjKG}_0, \mathsf{ProjKG}_1$). Then, for $\text{LOFC}_{\mathsf{full\text{-}aux}}$, we can use the hashing key $\mathsf{hk} = (\mathsf{hk}_0, \mathsf{hk}_1)$ and sets:

$$\mathsf{hp} \stackrel{\text{def}}{=} (\mathsf{hp}_0, \mathsf{hp}_1, H_\oplus) \quad \text{where } H_\oplus \stackrel{\text{def}}{=} \mathsf{Hash}_0(\mathsf{hk}_0, \mathsf{full\text{-}aux}, C) \oplus \mathsf{Hash}_1(\mathsf{hk}_1, \mathsf{full\text{-}aux}, C)$$

$$H \stackrel{\text{def}}{=} \mathsf{Hash}_0(\mathsf{hk}_0, \mathsf{full\text{-}aux}, C)$$

$$H' \stackrel{\text{def}}{=} \begin{cases} \mathsf{ProjHash}_0(\mathsf{hp}_0, \mathsf{full\text{-}aux}, C, r) & \text{if } C = \mathsf{Enc}(\mathsf{ek}, X_0; r) \\ H_\oplus \oplus \mathsf{ProjHash}_1(\mathsf{hp}_0, \mathsf{full\text{-}aux}, C, r) & \text{if } C = \mathsf{Enc}(\mathsf{ek}, X_1; r), \end{cases}$$

with $\oplus$ the exclusive or. Let us now show that the resulting protocol is not witness-indistinguishable, *i.e.*, a malicious verifier can know if the prover used X_0 or X_1 as witness. A malicious verifier can indeed pick $H_\oplus = 0$ (or a uniformly random value). Then an honest prover will send back $H = \mathsf{ProjHash}_0(\mathsf{hp}_0, \mathsf{full\text{-}aux}, C, r)$, if he encrypted $X = X_0$ in C and $H = 0 \oplus \mathsf{ProjHash}(\mathsf{hp}_1, \mathsf{full\text{-}aux}, C, r)$ otherwise. These two values are different with high probability, and so the malicious verifier can distinguish X_0 from X_1. We notice that this attack works whatever the encryption scheme is.

The previous problem does not happen for SPHFs where it is easy distinguish valid hp from invalid ones, such as for the SPHFs constructed in [BBC+13]. However, even in this case, we do not see how to prove that the resulting generic construction yields a zero-knowledge argument, because, if the simulator does not have access to hk, but only to hp, there is no trivial way to compute H.

7 Zero-Knowledge Arguments from TSPHFs

Let us now show how TSPHFs enable to construct E-ZK and true-simulation extractable zero-knowledge arguments (tSE-ZK). A tSE-ZK is a E-ZK in which extractability holds even if the adversary has access to simulated proofs for any valid statement. tSE-ZK is a relaxation of the notion of simulation-extractable zero-knowledge arguments[2], in which the adversary has access to simulated proofs of any statement (valid or invalid). But as shown in [Har11], tSE-ZK are sufficient for most applications, and can often be more efficient than SE-ZK. Formal definitions can be found in Appendix B.1.

7.1 Generic Construction

Let us now introduce our generic two-flow constructions for E-ZK and tSE-ZK, depicted in Figure 7. They are similar to the generic construction of HVE-ZK from SPHFs of Section 6.1, except the KV-SPHF is replaced by a KV-TSPHF and the verifier aborts if the received hp is not valid. Furthermore, the tSE-ZK version uses a labeled IND-CCA encryption scheme (instead of an IND-CPA encryption scheme) and the language of the KV-TSPHF has to be restricted to ciphertexts with the expected label ℓ.

It is also possible to use a GL-TSPHF (instead of a KV-TSPHF), at the expense of requiring three flows instead of two and using an additional one-time signature to prevent the adversary from mixing flows from different sessions (only for the tSE-ZK variant). The E-ZK three-flow variant is depicted in Figure 9, whereas tSE-ZK three-flow variant is depicted in Figure 9. The tSE-ZK three-flow variant requires a one-time signature (see Section A.3) to prevent the adversary from mixing flows. The parameters $\mathsf{param} \stackrel{\$}{\leftarrow} \mathsf{Setup}(1^\mathfrak{K})$ of the one-time signature are supposed to be in the CRS crs.

As for the construction of Section 6.1, completeness comes from the correctness of the SPHF and the extractor acts as an honest verifier and decrypts the ciphertext of the adversary. The simulator consists in encrypting an arbitrary value in C and computing H using hp and the trapdoor τ': $\mathsf{THash}(\mathsf{hp}, \mathsf{full\text{-}aux}, C, \tau')$. The

[2] Also called non-malleable arguments of knowledge when $y = \perp$ in [GMY06].

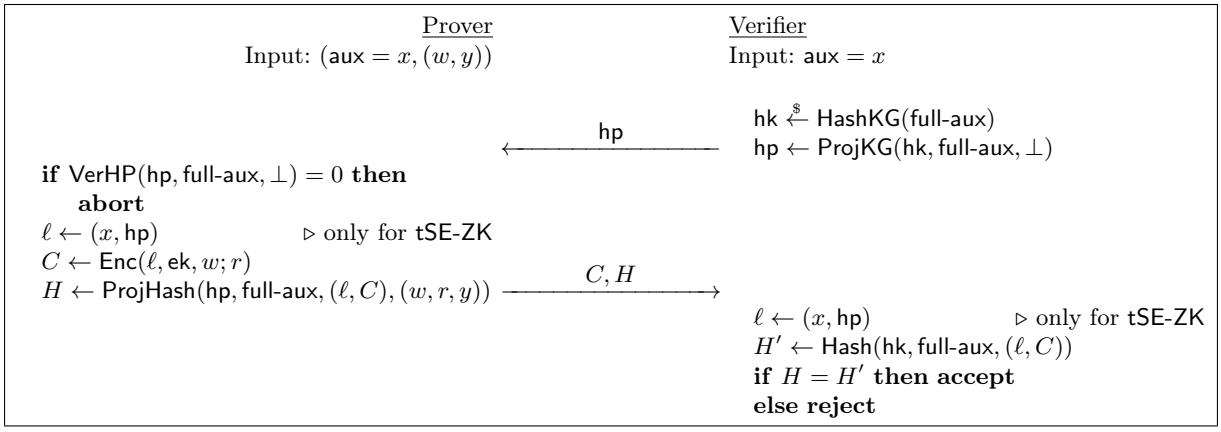

Fig. 7. E-ZK and tSE-ZK Arguments from KV-TSPHF (for the E-ZK version, the label ℓ is not used).

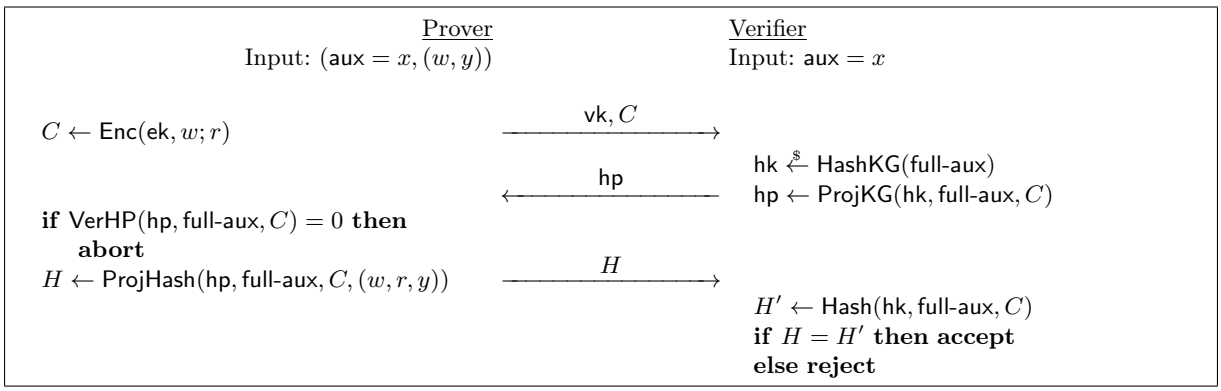

Fig. 8. E-ZK Argument from GL-TSPHFs.

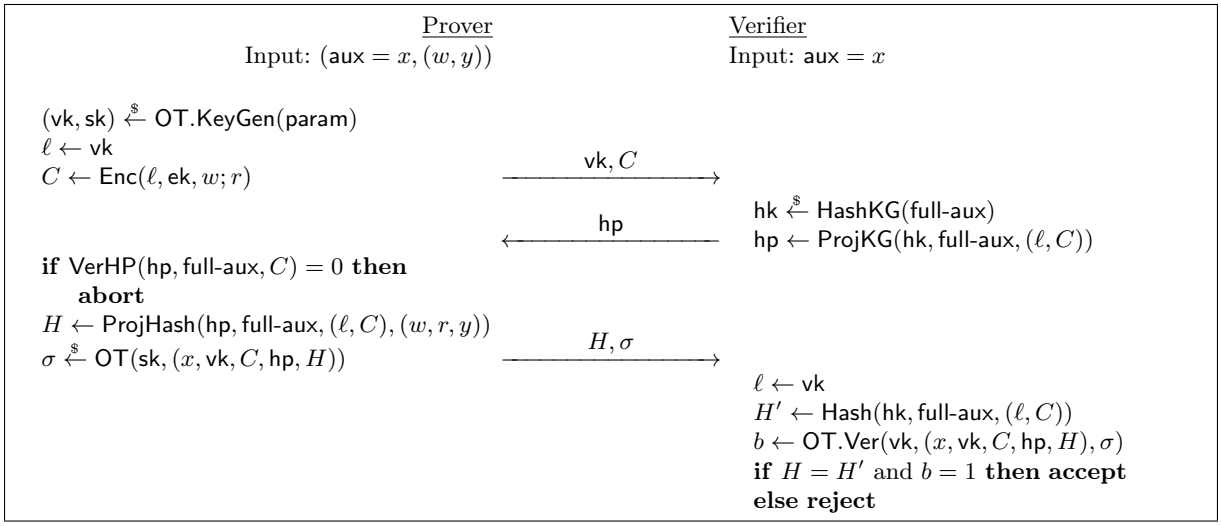

Fig. 9. tSE-ZK Argument from GL-TSPHFs.

IND-CPA property of the encryption scheme used for C ensures the simulator transcripts are computationally indistinguishable from real transcripts, which proves the zero-knowledge property of our constructions.

Soundness, extractability and true-simulation extractability (for the tSE-ZK variant) are slightly more complex. For them to be true, we require that, for any w and x, knowing τ provides a way to test whether x is valid and w is a partial witness of x, with overwhelming probability. This property is actually always verified by TSPHF constructed as in Section 4.3, as shown in Appendix F.2. In addition, the true-simulation extractability requires the IND-CCA property of the labeled encryption scheme (contrary to all previous properties). Proofs are given in Appendix D.2.

7.2 An Instantiation

The KV-SPHF and the HVE-ZK for multi-exponentiation equations in Section 6.1 and Appendix C.1 can directly be transformed into a KV-TSPHF and an E-ZK, respectively (supposing $\mathbb{G} = \mathbb{G}_1$ where $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is an asymmetric bilinear group, where the DDH assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$), if all constants $A_{k,j}$ are in crs, and crs be generated in such a way their discrete logarithm is known. It is important to notice that neither the simulator nor the extractor uses τ, which contains these discrete logarithms. These discrete logarithms are only required in the proof of security of the soundness and of the extractability.

We remark that for the used KV-TSPHF, the hash value H is in $\mathbb{G}_T$. To avoid sending such a large element, instead of sending H, the prover can extract the entropy of H and send to the prover the result of the extraction (instead of H), which is smaller than an element of $\mathbb{G}_1$ and so considered as an element of $\mathbb{G}_1$, in the comparison below.

Let us now analyze the transmission complexity of our constructions. The E-ZK just requires $t+1$ more elements in $\mathbb{G}_2$ for hp (with ElGamal ciphertexts with reuse of randomness). Therefore it requires $t+1$ more elements in $\mathbb{G}_2$ and $t+1$ fewer elements in $\mathbb{G}_1$ or $\mathbb{Z}_p$ than the corresponding Ω-protocol, but is zero-knowledge instead of being just honest-verifier zero-knowledge. And it still requires fewer elements or about the same number of elements (depending on the exact equations) than Groth-Sahai NIZK.

In addition, if, in the above E-ZK protocol, the ElGamal encryption scheme is replaced by a Cramer-Shoup encryption scheme with reuse of randomness, we get a tSE-ZK. The full scheme can be found in Appendix C.3. We remark that, compared to the E-ZK scheme, the tSE-ZK scheme just requires 2 elements (in $\mathbb{G}_1$) more for the ciphertext C, and 1 element more in $\mathbb{G}_1$ and 4 more in $\mathbb{G}_2$ for hp. And therefore, its transmission complexity is still comparable to the corresponding Ω-protocol, and still better or comparable than Groth-Sahai NIZK, in most cases, while being tSE-ZK instead of only E-ZK.

8 Verifiable Encryption of Waters Signatures

In this section, we show a concrete instantiation of our constructions of HVE-ZK and E-ZK of Sections 6.2 and 7.2, for the language of ciphertexts of valid Waters signatures in an asymmetric bilinear group (defined in Section A.4). These constructions can be used, for example, in optimistic fair exchanges of signatures [ASW98]. We remark that, for this language $w = \perp$, and so tSE-ZK are equivalent to E-ZK.

We first precisely describe the language we are interested in, and then describe the obtained HVE-ZK and E-ZK protocols and compare them to Groth-Sahai NIZK and Σ-protocols.

8.1 Language $\mathcal{L}$

Let us suppose crs contains $\mathsf{ek} = (g_{1,1}, g_{1,2}, c, d, h, \mathfrak{H}_K)$ a public key for the Cramer-Shoup encryption scheme (in $\mathbb{G}_1$), and $\mathsf{param} = (g_1, g_2, \boldsymbol{f}, \mathfrak{h})$ parameters for the Waters signature scheme. The trapdoor τ will contain the discrete logarithm of all the elements of ek and param in base g_1 for elements in $\mathbb{G}_1$ and in base g_2 for elements in $\mathbb{G}_2$, and so, in particular the decryption key dk can be computed from τ. It is important to notice that the encryption key ek is not the one we previously used to encrypt w (in C) in the generic HVE-ZK construction from Section 6, since $w = C = \perp$.

Let us now consider the following language of words $x = (\ell, \mathsf{vk}, M, E_1, \boldsymbol{\sigma_2})$ where:

- ℓ is a label;
- $\mathsf{vk} = (g_1^z, g_2^z)$ is a public key for the Waters signature scheme[3];
- M is a message in $\{0,1\}^k$;
- $E_1 = (\mathbf{u} = (u_1 = g_{1,1}^r, u_2 = g_{1,2}^r), e = h^r \cdot \sigma_1, v = (cd^\xi)^r)$ is a Cramer-Shoup encryption under label ℓ of some element $\sigma_1 \in \mathbb{G}_1$ ($\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$);
- $\boldsymbol{\sigma_2} = (\sigma_{2,1}, \sigma_{2,2}) \in \mathbb{G}_1 \times \mathbb{G}_2$;

such that: $\sigma = (\sigma_1, \boldsymbol{\sigma_2})$ is a valid Waters signature of M under vk, *i.e.*, $e(\sigma_1, g_2) = e(\mathfrak{h}, \mathsf{vk}_2) \cdot e(\mathcal{F}(M), \sigma_{2,2})$, and $e(\sigma_{2,1}, g_2) = e(g_1, \sigma_{2,2})$.

We do not encrypt $\boldsymbol{\sigma_2}$, since it is not necessary in most application, because Waters signature can be randomized in such a way $\boldsymbol{\sigma_2}$ is a completely random ordered pair, such that $e(\sigma_{2,1}, g_2) = e(g_1, \sigma_{2,2})$. We remark that, anyone can check the condition $e(\sigma_{2,1}, g_2) = e(g_1, \sigma_{2,2})$ without knowing any secret value and so it is not necessary to check this condition in the language $\mathcal{L}$ of the HVE-ZK or E-ZK. Furthermore, the condition $e(\sigma_1, g_2) = e(\mathfrak{h}, \mathsf{vk}_2) \cdot e(\mathcal{F}(M), \sigma_{2,2})$, can be rewritten

$$\exists z, s,\ \mathsf{vk}_1 = g_1^z,\ \sigma_{2,1} = g_1^s, \text{ and } \sigma_1 = \mathfrak{h}^z \cdot \mathcal{F}(M)^s,$$

since $\mathsf{vk}_1 = g_1^z$ and $\mathsf{vk}_2 = g_2^z$ for some z. This rewriting avoids using pairing and is of the form of a system of multi-exponentiation equations (in $\mathbb{G}_1$) as defined in Section 6.2.

Notice this rewriting requires the prover to know z and s, which is not a problem in applications such as optimistic fair exchange, where the Waters signature is issued by the user who encrypts it and proves it has been encrypted correctly.

Let us now formally show the resulting language $\mathcal{L}$:

$$\exists r, z, s,\ \begin{cases} u_1 = g_{1,1}^r,\ u_2 = g_{1,2}^r, \text{ and } v = (cd^\xi)^r \\ \mathsf{vk}_1 = g_1^z,\ \sigma_{2,1} = g_1^s, \text{ and } e = h^r \cdot \mathfrak{h}^z \cdot \mathcal{F}(M)^s \end{cases}$$

which is also of the form of a system of multi-exponentiation equations (in $\mathbb{G}_1$). Therefore, we can use HVE-ZK and E-ZK instantiations of Sections 6.2 and 7.2.

8.2 HVE-ZK

In this section, we completely write down the SPHF used in the HVE-ZK construction, and also the corresponding Σ-protocol (which is the classical way to do such a HVE-ZK).

SPHF. Since $w = \perp$, $C = \perp$ and we can simplify the construction of Appendix C.1 and choose:

$$\Gamma = \begin{pmatrix} g_{1,1} & g_{1,2} & cd^\xi & 1 & 1 & h \\ 1 & 1 & 1 & g_1 & 1 & \mathfrak{h} \\ 1 & 1 & 1 & 1 & g_1 & \mathcal{F}(M) \end{pmatrix} \qquad \begin{aligned} \Theta_{\mathsf{aux}} &= (u_1, u_2, v, \mathsf{vk}_1, \sigma_{2,1}, e) \\ \boldsymbol{\lambda} &= (r, z, s) \\ \boldsymbol{\lambda} \cdot \Gamma &= \left(g_{1,1}^r, g_{1,2}^r, (cd^\xi)^r, g_1^z, g_1^s, h^r \cdot \mathfrak{h}^z \cdot \mathcal{F}(M)^s\right), \end{aligned}$$

which yields the following projection keys and hash value:

$$\begin{aligned} \mathsf{hp} &= (g_{1,1}^{\alpha_1} \cdot g_{1,2}^{\alpha_2} \cdot (cd^\xi)^{\alpha_3} \cdot h^{\alpha_6}, g_1^{\alpha_4} \cdot \mathfrak{h}^{\alpha_6}, g_1^{\alpha_5} \cdot \mathcal{F}(M)^{\alpha_6}) \\ H &= u_1^{\alpha_1} \cdot u_2^{\alpha_2} \cdot v^{\alpha_3} \cdot \mathsf{vk}_1^{\alpha_4} \cdot \sigma_{2,1}^{\alpha_5} \cdot e^{\alpha_6} = \mathsf{hp}_1^r \cdot \mathsf{hp}_2^z \cdot \mathsf{hp}_3^s = H'. \end{aligned}$$

We remark that the resulting SPHF has only one word: $C = \perp$ ($\mathcal{S}et = \{\perp\}$). So the SPHF is just used to prove that $\mathsf{aux} = x = (\ell, \mathsf{vk}, M, E_1, \boldsymbol{\sigma_2})$ is "valid": if aux is valid, $C = \perp$ is valid, and otherwise $C = \perp$ is invalid.

[3] The public key is supposed to be valid, *i.e.*, $(g_1, g_2, \mathsf{vk}_1, \mathsf{vk}_2)$ is supposed to be a valid DDH tuple. This can be verified using a pairing.

Σ-Protocol. An Σ-protocol is a three-flow honest-verifier extractable, where the prover first sends a message com called commitments, then the verifier sends back a message $\mathfrak{C}$ called challenge, and finally the prover answers with a message resp called response. Notice that $w = \perp$, and so we do not need a Ω-protocol for the language $\mathcal{L}$ of Cramer-Shoup ciphertexts of Waters signature. Details on Σ-protocols and Ω-protocols can be found in B.2.

Here is the complete Σ-protocol.

- commitment: $\mathsf{com} = (u'_1, u'_2, v', \mathsf{vk}'_1, \sigma'_{2,1}, e')$, where $u'_1 = g_{1,1}^{r'}$, $u'_2 = g_{1,2}^{r'}$, $v' = (cd^{\xi})^{r'}$, $\mathsf{vk}'_1 = g_1^{z'}$, $\sigma'_{2,1} = g_1^{s'}$ and $e' = h^{r'} \cdot \mathfrak{h}^{z'} \cdot \mathcal{F}(M)^{s'}$ with $r', z', s' \xleftarrow{\$} \mathbb{Z}_p$;
- challenge: $\mathfrak{C} \xleftarrow{\$} \mathbb{Z}_p$;
- response: $\mathsf{resp} = (r'', z'', s'')$ with $r'' = r' + \mathfrak{C}r \bmod p$, $z'' = z' + \mathfrak{C}z \bmod p$ and $s'' = s' + \mathfrak{C}s \bmod p$;
- verification:

$$u_1^{\mathfrak{C}} \cdot u'_1 = g_{1,1}^{r''} \qquad u_2^{\mathfrak{C}} \cdot u'_2 = g_{1,2}^{r''} \qquad v^{\mathfrak{C}} \cdot v' = (cd^{\xi})^{r''}$$
$$\mathsf{vk}_1^{\mathfrak{C}} \cdot \mathsf{vk}'_1 = g_1^{z''} \qquad \sigma_{2,1}^{\mathfrak{C}} \cdot \sigma'_{2,1} = g_1^{s''} \qquad e^{\mathfrak{C}} \cdot e' = h^{r''} \cdot \mathfrak{h}^{z''} \cdot \mathcal{F}(M)^{s''}.$$

8.3 E-ZK.

Let us completely write down the TSPHF for the E-ZK construction. The projection hash and the hash values are:

$$\mathsf{hp} = (g_{1,1}^{\alpha_1} \cdot g_{1,2}^{\alpha_2} \cdot (cd^{\xi})^{\alpha_3} \cdot h^{\alpha_6}, g_1^{\alpha_4} \cdot \mathfrak{h}^{\alpha_6}, g_1^{\alpha_5} \cdot \mathcal{F}(M)^{\alpha_6}, \chi_1 = \zeta^{\alpha_1}, \ldots, \chi_6 = \zeta^{\alpha_6})$$
$$H = e(u_1^{\alpha_1} \cdot u_2^{\alpha_2} \cdot v^{\alpha_3} \cdot \mathsf{vk}_1^{\alpha_4} \cdot \sigma_{2,1}^{\alpha_5} \cdot e^{\alpha_6}, g_2) = e(\mathsf{hp}_1^r \cdot \mathsf{hp}_2^z \cdot \mathsf{hp}_3^s, g_2) = H',$$

where $\tau' \xleftarrow{\$} \mathbb{Z}_p$ and $\mathsf{crs}' = \zeta = g_2^{\tau'} \xleftarrow{\$} \mathbb{G}_2$. The projection key is valid if and only if:

$$e(\mathsf{hp}_1, \zeta) = e(g_{1,1}, \chi_1) \cdot e(g_{1,2}, \chi_2) \cdot e(cd^{\xi}, \chi_3) \cdot e(h, \chi_6)$$
$$e(\mathsf{hp}_2, \zeta) = e(g_1, \chi_4) \cdot e(\mathfrak{h}, \chi_6)$$
$$e(\mathsf{hp}_3, \zeta) = e(g_1, \chi_5) \cdot e(\mathcal{F}(M), \chi_6)$$

and the hash value can also be computed as:

$$H = (e(u_1, \chi_1) \cdot e(u_2, \chi_2) \cdot e(v, \chi_3) \cdot e(\mathsf{vk}_1, \chi_4) \cdot e(\sigma_{2,1}, \chi_5) \cdot e(e, \chi_6))^{\tau'^{-1}} = H''.$$

8.4 Comparison.

In Table 1, we compare the transmission complexity of our constructions of HVE-ZK and E-ZK from SPHFs and TSPHFs to corresponding Σ-protocol and Groth-Sahai NIZK. The cost of Groth-Sahai NIZK is computed from tables given in [GS08] (the 6 elements in $\mathbb{G}_2$ are for commitments of r, z and s, whereas the 6 elements in $\mathbb{Z}_p$ are for the 6 linear multi-exponentiation equations in $\mathbb{G}_1$).

We recall that for the E-ZK version, H is an element of $\mathbb{G}_T$, and that the prover does not send H but uses a randomness extractor (as explained in Section 7.2). The resulting element is smaller than an element of $\mathbb{G}_1$ and we count it as an element of $\mathbb{G}_1$ in our comparison.

Our HVE-ZK outperforms the Σ-protocol and the Groth-Sahai NIZK. Our E-ZK outperforms the Groth-Sahai NIZK and is still competitive with the Σ-protocol, though being a two-flow E-ZK instead of only a three-flow HVE-ZK.

Table 1. Transmission complexity of arguments for encryption of a Waters signature.

	$\mathbb{Z}_p$	$\mathbb{G}_1$	$\mathbb{G}_2$
HVE-ZK from SPHFs	0	4	0
E-ZK from SPHFs	0	4	6
Σ-protocol (HVE-ZK)	4	6	0
Groth-Sahai NIZK (E-ZK)	6	0	6

References

ACP09. Michel Abdalla, Céline Chevalier, and David Pointcheval. Smooth projective hashing for conditionally extractable commitments. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 671–689. Springer, August 2009.

ASW98. N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures (extended abstract). In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 591–606. Springer, May / June 1998.

BBC+13. Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New smooth projective hash functions and one-round authenticated key exchange. Cryptology ePrint Archive, Report 2013/034, 2013. `http://eprint.iacr.org/`.

BBS03. Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. Randomness re-use in multi-recipient encryption schemeas. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 85–99. Springer, January 2003.

BBS04. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, August 2004.

BFPV11. Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 403–422. Springer, March 2011.

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

BPR00. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer, May 2000.

BPV12. Olivier Blazy, David Pointcheval, and Damien Vergnaud. Round-optimal privacy-preserving protocols with smooth projective hash functions. In *TCC 2012: 9th Theory of Cryptography Conference*, Lecture Notes in Computer Science, pages 94–111. Springer, 2012.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145. IEEE Computer Society Press, October 2001.

CHK+05. Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421. Springer, May 2005.

CKS11. Jan Camenisch, Stephan Krenn, and Victor Shoup. A framework for practical universally composable zero-knowledge protocols. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 449–467. Springer, December 2011.

CS98. Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, August 1998.

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

ElG85. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, August 1985.

GGH12. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices and applications. Cryptology ePrint Archive, Report 2012/610, 2012. `http://eprint.iacr.org/`.

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

GMY06. Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, April 2006.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, April 2008.

Har11. Kristiyan Haralambiev. *Efficient Cryptographic Primitives for Non-Interactive Zero-Knowledge Proofs and Applications*. PhD thesis, New York University, 2011.

HK07. Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 553–571. Springer, August 2007.

JR12. Charanjit S. Jutla and Arnab Roy. Relatively-sound NIZKs and password-based key-exchange. In *PKC 2012: 15th International Workshop on Theory and Practice in Public Key Cryptography*, Lecture Notes in Computer Science, pages 485–503. Springer, 2012.

KOY01. Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 475–494. Springer, May 2001.

KV11. Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 293–310. Springer, March 2011.

Sch91. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

Sha07. Hovav 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/2007/074.pdf`.

A Preliminaries

This appendix reviews the classical notations used in this paper, with the concrete ElGamal and Cramer-Shoup encryption schemes, secure under the DDH assumption, and the Waters signature.

A.1 Statistical Distance

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two probability distributions over a finite set $\mathcal{S}$ and let X and Y be two random variables with these two respective distributions. The statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is also the statistical distance between X and Y:

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

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

A.2 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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$ is a bilinear map (called a pairing) with the following properties:

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

It is called a *symmetric* bilinear 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.

A.3 Formal Definition of the Primitives

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

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

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

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

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,

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

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

  - $\mathsf{ODec}(\ell, c)$: 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 $\mathcal{CT}$.

  The advantages are

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

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

$\mathsf{Exp}^{\mathsf{ind\text{-}cca}-b}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$
$\mathsf{param} \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
$(\ell^*, m_0, m_1) \leftarrow \mathcal{A}(\mathtt{FIND} : \mathsf{ek}, \mathsf{ODec}(\cdot,\cdot))$
$c^* \leftarrow \mathsf{Enc}(\ell^*, \mathsf{ek}, m_b)$
$b' \leftarrow \mathcal{A}(\mathtt{GUESS} : c^*, \mathsf{ODec}(\cdot,\cdot))$
if $(\ell^*, c^*) \in \mathcal{CT}$ **then return** 0
else return b'

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 ODec access:

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

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

Signature Scheme and One-Time Signature Scheme. A signature scheme is defined by four algorithms:

- $\mathsf{Setup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}(\mathsf{param})$ generates a pair of keys, the verification key vk and the signing key sk;
- $\mathsf{Sign}(\mathsf{sk}, m; s)$ produces a signature σ on the input message m, under the signing key sk, and using the random coins s;
- $\mathsf{Ver}(\mathsf{vk}, m, \sigma)$ checks whether σ is a valid signature on m, w.r.t. the public key vk; it outputs 1 if the signature is valid, and 0 otherwise.

A signature scheme $\mathcal{S}$ should satisfy the following properties

- *Correctness*: for all key pair $(\mathsf{vk}, \mathsf{sk})$, all random coins s and all messages m, $\mathsf{Ver}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m; s)) = 1$.
- *Existential unforgeability under (adaptive) chosen-message attacks*: this security notion can be formalized by the following security game, where it makes use of the oracle OSign:

  - $\mathsf{OSign}(m)$: This oracle outputs a valid signature on m under the signing key sk. The input queries m are added to the list $\mathcal{SM}$.

  The success probabilities are

$$\mathsf{Succ}^{\mathsf{euf-cma}}_{\mathcal{S}}(\mathcal{A}) = \Pr[\mathsf{Exp}^{\mathsf{euf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K}) = 1] \qquad \mathsf{Succ}^{\mathsf{euf-cma}}_{\mathcal{S}}(t, q_s) = \max_{\mathcal{A} \le t, q_s} \{\mathsf{Succ}^{\mathsf{euf-cma}}_{\mathcal{S}}(\mathcal{A})\},$$

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

$\mathsf{Exp}^{\mathsf{euf-cma}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K})$
$\mathsf{param} \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
$(m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{vk}, \mathsf{OSign}(\cdot))$
$b \leftarrow \mathsf{Ver}(\mathsf{vk}, m^*, \sigma^*)$
if $M \in \mathcal{SM}$ **then return** 0
else return b

A one-time signature scheme $(\mathsf{OT.Setup}, \mathsf{OT.KeyGen}, \mathsf{OT.Sign}, \mathsf{OT.Ver})$ is similar to a signature scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ except that, in the experiment for existential unforgeability, the adversary is allowed to do at most one signature query to OSign: $q_s \le 1$.

A.4 Concrete Instantiations

In the body of this paper, we focus on the sole decisional Diffie-Hellman (DDH) assumption:

Definition 1 (Decisional Diffie-Hellman (DDH)). *The Decisional Diffie-Hellman assumption says that, in a group $(p, \mathbb{G}, g)$, when we are given (g^a, g^b, g^c) for unknown random $a, b \xleftarrow{\$} \mathbb{Z}_p$, it is hard to decide whether $c = ab \bmod p$ (a DH tuple) or $c \xleftarrow{\$} \mathbb{Z}_p$ (a random tuple). We define by $\mathsf{Adv}^{\mathsf{ddh}}_{p,\mathbb{G},g}(t)$ the best advantage an adversary can have in distinguishing a DH tuple from a random tuple within time t.*

For asymmetric bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, we also use the SXDH assumption which states that the DDH assumption holds both in $\mathbb{G}_1$ and in $\mathbb{G}_2$

IND-CPA Encryption: ElGamal. The ElGamal encryption scheme [ElG85] is defined as follows:

- $\mathsf{Setup}(1^{\mathfrak{K}})$ generates a group $\mathbb{G}$ of order p, with a generator g;
- $\mathsf{KeyGen}(\mathsf{param})$ generates $\mathsf{dk} = z \xleftarrow{\$} \mathbb{Z}_p$, and sets, $\mathsf{ek} = h = g^z$;
- $\mathsf{Enc}(\mathsf{ek}, M; r)$, for a message $M \in \mathbb{G}$ and a random scalar $r \in \mathbb{Z}_p$, the ciphertext is $C = (u = g^r, e = M \cdot h^r)$;
- $\mathsf{Dec}(\mathsf{dk}, C)$: one computes $M = e/u^z$ and outputs M.

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

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

Randomness-Reuse. When one wants to encrypt vectors of messages, one can concatenate independent ciphertexts, with possibly a common ξ for the Cramer-Shoup encryption scheme in order to keep the global non-malleability. But for efficiency considerations, one can also re-use random coins [BBS03]: if one wants to encrypt $(M_i)_i \in \mathbb{G}^\ell$, where i ranges from 1 to ℓ,

- for the ElGamal encryption scheme, if one chooses $\mathsf{dk} = (z_i)_i$, and sets, $\mathsf{ek} = (h_i = g^{z_i})_i$, the encryption of $(M_i)_i$ can be done as $\mathsf{Enc}(\mathsf{ek}, (M_i)_i; r) = (u = g^r, (e_i = M_i \cdot h_i^r)_i)$;
- for the Cramer-Shoup encryption scheme, if one chooses $\mathsf{dk} = (x_1, x_2, y_1, y_2, (z_i)_i)$, and sets $\mathsf{ek} = (g_1, g_2, c, d, (h_i = g_1^{z_i})_i, \mathfrak{H}_K)$ as above except for $(h_i)_i$, the encryption of $(M_i)_i$ can be done as $\mathsf{Enc}(\ell, \mathsf{ek}, (M_i)_i; r) = (\mathbf{u} = (g_1^r, g_2^r), (e_i = M_i \cdot h_i^r)_i, v = (cd^\xi)^r)$, where v is computed afterwards with $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, (e_i)_i)$.

This is much more compact: for ElGamal, the ciphertext consists of $\ell+1$ group elements instead of 2ℓ, whereas for Cramer-Shoup, the ciphertext consists of $\ell+3$ group elements instead of 4ℓ, but still with the same security level: $\mathsf{IND\text{-}CPA}$ for the former, and $\mathsf{IND\text{-}CCA}$ for the latter, both under the DDH assumption.

Waters Signature in Asymmetric Bilinear Groups. In Section 8, we use the Waters signature scheme for asymmetric bilinear group, which has been proposed and proved in [BFPV11]:

- $\mathsf{Setup}(1^{\mathfrak{K}})$: generates a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T)$, and chooses a random vector $\boldsymbol{f} = (f_0, \ldots, f_k) \xleftarrow{\$} \mathbb{G}_1^{k+1}$ that defines the Waters hash function $\mathcal{F}(M) = f_0 \prod_{i=1}^k f_i^{M_i}$ for $M \in \{0,1\}^k$, and an extra generator $\mathfrak{h} \xleftarrow{\$} \mathbb{G}_1$. The global parameters param consists of $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, \boldsymbol{f}, \mathfrak{h})$;
- $\mathsf{KeyGen}(\mathsf{param})$ chooses a random scalar $z \xleftarrow{\$} \mathbb{Z}_p$, which defines the public $\mathsf{vk} = (g_1^z, g_2^z)$, and the secret key $\mathsf{sk} = \mathfrak{h}^z$;
- $\mathsf{Sign}(\mathsf{sk}, M; s)$ outputs, for some random $s \xleftarrow{\$} \mathbb{Z}_p$, $\sigma = (\sigma_1 = \mathsf{sk} \cdot \mathcal{F}(M)^s, \boldsymbol{\sigma_2} = (g_1^s, g_2^s))$;
- $\mathsf{Ver}(\mathsf{vk}, M, \sigma)$ checks whether $e(\sigma_1, g_2) = e(\mathfrak{h}, \mathsf{vk}_2) \cdot e(\mathcal{F}(M), \sigma_{2,2})$, and $e(\sigma_{2,1}, g_2) = e(g_1, \sigma_{2,2})$.

This scheme is unforgeable under the following variant of the CDH assumption:

Definition 2 (The Advanced Computational Diffie-Hellman problem (CDH^+)). *In an asymmetric bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T)$. The CDH^+ assumption states that given $(g_1, g_2, g_1^a, g_2^a, g_1^b)$, for random $a, b \in \mathbb{Z}_p$, it is hard to compute g_1^{ab}.*

B Zero-Knowledge Arguments

B.1 Zero-Knowledge Arguments

In this section, we give some formal definitions for partially-extractable zero-knowledge arguments, using the formalism of Garay-MacKenzie-Yang (GMY) in [GMY06], after briefly recalling this formalism. These definitions are simple adaptations of classical definitions.

GMY Formalism. For two probabilistic interactive Turing machines (ITM) A and B, $\langle A, B\rangle_\sigma(\mathsf{in})$ is the local output of B after an interactive execution with A using CRS σ and common input in. The transcript tr of a machine is a tuple composed of its common input in, the messages received on its input tape and the messages sent through its output tape. If the ordered input messages of a transcript tr correspond to the ordered output messages of a transcript tr' and vice versa, we say that tr and tr' *match* and we write $\mathsf{tr} \bowtie \mathsf{tr}'$. If tr and tr' do not match, we write $\mathsf{tr} \not\bowtie \mathsf{tr}'$.

For any ITM A, we also denote by $\boxed{A}$ its multi-session extension or protocol wrapper. $\boxed{A}$ works as follows:

- on input message $(\mathsf{START}, \ell, \mathsf{in}, \mathsf{priv})$, $\boxed{A}$ starts a new interactive machine A with label ℓ, common input in, private input priv and fresh random tape;
- on input message (MSG, ℓ, m), $\boxed{A}$ sends the message m to the interactive machine with label ℓ (if it exists), and returns the output message of this machine.

All machines A started by $\boxed{A}$ use the same CRS σ.

Let $\boxed{A}_1$ be the single-session extension of A, which works as $\boxed{A}$, except it only accepts one START query. The output of $\boxed{A}_1$ is the tuple $(\mathsf{in}, \mathsf{tr}, v)$ where in is the common input, tr is the transcript of the machine A started by $\boxed{A}_1$ and v is the output of A. The output of $\boxed{A}$ is a tuple $(\boldsymbol{\mathsf{in}}, \boldsymbol{\mathsf{tr}}, \boldsymbol{v})$ of three vectors, such that $(\mathsf{in}_i, \mathsf{tr}_i, v_i)$ is the tuple that $\boxed{A}_1$ would have output for the ith machine started by $\boxed{A}$.

Two ITM B and C are said to be *coordinated* if they have a single control (and, in particular, a common state), but two distinct sets of input/output communication tapes. For four interactive Turing machines A, B, C and D, with B and C coordinated, $(\langle A, B\rangle, \langle C, D\rangle)_\sigma$ is the local output of D after an interactive execution with C and an interactive execution between A and B, all using the CRS σ.

(Partially Extractable) Zero-Knowledge Arguments (E-ZK). An (unbounded partially extractable) zero-knowledge argument (E-ZK) system for a witness relation $\mathcal{R}$ (defined as in Section 1) is a tuple $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2), \mathsf{Ext})$, where:

- Setup is a probabilistic polynomial-time TM (PPT) which takes the security parameter $\mathfrak{K}$ in unary as input and outputs a CRS σ;
- Prove is a polynomial-time ITM which takes a statement $x \in \mathcal{L}$ as common input and a valid witness (w, y) as private input (*i.e.*, $\mathcal{R}(x, (w, y)) = 1$) and is able to run a protocol (with a verifier Ver) to prove that $x \in \mathcal{L}$;
- Ver is a polynomial-time ITM which takes a statement x as common input, is able to run a protocol (with a prover Prove) and outputs 1 if it accepts the proof of the prover and 0 otherwise;
- Sim_1 is a PPT which takes the security parameter $\mathfrak{K}$ as input and outputs a simulated CRS σ together with a trapdoor τ;
- Sim_2 is a polynomial-time ITM which takes the trapdoor τ as private input and a statement x as common input, and is able to simulate a run of Prove (without knowing w nor y);
- Ext is a polynomial-time ITM which takes as private input the trapdoor τ and as common input a statement x, and is able to simulate a run of Ver in such a way that, if Ver accepts, it is able to extract a valid partial witness w for x. Ext outputs a pair (b, w) where b is the status of the statement and w is the witness;

such that the following properties are verified:

- *Completeness.* Π is ε-complete, if for all $x \in \mathcal{L}$ and all valid witnesses (w, y) of x:

$$\Pr\left[\sigma \xleftarrow{\$} \mathsf{Setup}(1^{\mathfrak{K}}) \,;\, \langle \mathsf{Prove}((w, y)), \mathsf{Ver}\rangle_\sigma(x) = 1\right] \geq 1 - \varepsilon;$$

– *Soundness.* Π is (t,ε)-sound, if for all adversary $\mathcal{A}$ running in time at most t, and for all words $x \notin \mathcal{L}$:

$$\Pr\left[\sigma \xleftarrow{\$} \mathsf{Setup}(1^{\mathfrak{K}})\ ;\ \langle \mathcal{A}, \mathsf{Ver}\rangle_{\sigma}(x) = 1\right] \leq \varepsilon;$$

– *Reference String Indistinguishability.* Π is (t,ε)-reference-string-indistinguishable if for any adversary $\mathcal{A}$ running in time at most t:

$$\left|\Pr\left[\sigma \xleftarrow{\$} \mathsf{Setup}(1^{\mathfrak{K}})\ ;\ \mathcal{A}(\sigma) = 1\right] - \Pr\left[\sigma \xleftarrow{\$} \mathsf{Sim}_1(1^{\mathfrak{K}})\ ;\ \mathcal{A}(\sigma) = 1\right]\right| = 1;$$

– *Extractor Indistinguishability.* Π is extractor-indistinguishable if, for any $\tau \in \{0,1\}^*$, for any (unbounded) adversary $\mathcal{A}$, the distribution of $\langle \mathcal{A}, \boxed{\mathsf{Ver}}_1 \rangle$ is identical to the distribution of $\langle \mathcal{A}, \boxed{\mathsf{Ext}_1(\tau)}_1 \rangle$ when we restrict the output of Ext_1 to the first element b of the ordered pair (b,w);

– *(Partial) Extractability.* Π is (t,ε)-(partially)-extractable if, for any adversary $\mathcal{A}$ running in time at most t, $\Pr\left[\mathsf{Exp}^{\mathrm{ext}}_{\mathcal{A}}(\mathfrak{K}) = 1\right] \leq \varepsilon$, where the experiment $\mathsf{Exp}^{\mathrm{ext}}_{\mathcal{A}}$ is defined as follows:

$\mathsf{Exp}^{\mathrm{ext}}_{\mathcal{A}}(\mathfrak{K})$
  $(\sigma,\tau) \xleftarrow{\$} \mathsf{Sim}_1(1^{\mathfrak{K}})$
  $(x, \mathsf{tr}, (b,w)) \xleftarrow{\$} \langle \mathcal{A}, \boxed{\mathsf{Ext}(\tau)}_1 \rangle_{\sigma}$
  if $b = 1$ and $\forall y,\ \mathcal{R}(x,(w,y)) = 0$ **then**
    return 1
  else
    return 0

– *(Unbounded) Zero-Knowledge.* Π is (t,ε)-(unbounded)-zero-knowledge if, for any adversary $\mathcal{A}$ running in time at most t, $|\Pr\left[\mathsf{Exp}^{\text{zk-0}}_{\mathcal{A}}(\mathfrak{K}) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{zk-1}}_{\mathcal{A}}(\mathfrak{K}) = 1\right]| \leq \varepsilon$, where the experiments $\mathsf{Exp}^{\mathrm{zk}-0}_{\mathcal{A}}$ and $\mathsf{Exp}^{\mathrm{zk}-1}_{\mathcal{A}}$ are defined as follows:

$\mathsf{Exp}^{\mathrm{zk}-0}_{\mathcal{A}}(\mathfrak{K})$
  $\sigma \xleftarrow{\$} \mathsf{Setup}(1^{\mathfrak{K}})$
  return $\langle \boxed{\mathsf{Prove}}, \mathcal{A} \rangle_{\sigma}$

$\mathsf{Exp}^{\mathrm{zk}-1}_{\mathcal{A}}(\mathfrak{K})$
  $(\sigma,\tau) \xleftarrow{\$} \mathsf{Sim}_1(1^{\mathfrak{K}})$
  return $\langle \boxed{\mathsf{Sim}'(\tau)}, \mathcal{A} \rangle_{\sigma}$

where $\mathsf{Sim}'(\tau)$ takes as common input a statement x and as private input a witness (w,y), runs $\mathsf{Sim}_2(\tau)$ with common input x if $\mathcal{R}(x,(w,y)) = 1$ and aborts otherwise. We remark this means the zero-knowledge property only holds for valid statements.

We remark that the soundness is implied by the partial extractability, while the extractor indistinguishability and the reference string indistinguishability are implied by the zero-knowledge property. However, we keep the soundness for simplicity and we keep the reference string indistinguishability, because it is useful for weaker variants of E-ZK, when the zero-knowledge property is no more enforced.

We often forget mentioning the extractor indistinguishability and the reference string indistinguishability in the proofs, for the sake of simplicity. For example, when we say that some property comes from the partial extractability, it means it comes from the partial extractability, the extractor indistinguishability and the reference string indistinguishability.

Honest-Verifier Zero-Knowledge Arguments (HVE-ZK). An honest-verifier zero-knowledge argument (HVE-ZK) system for a witness relation $\mathcal{R}$ is a tuple $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2), \mathsf{Ext})$, which verifies the same properties as an E-ZK, except the zero-knowledge property is replaced by the following weaker property:

– *Honest-Verifier Zero-Knowledge.* Π is (t, ε)-honest-verifier-zero-knowledge if, for any adversary $\mathcal{A}$ running in time at most t, $|\Pr\left[\mathsf{Exp}^{\text{hvzk-0}}_{\mathcal{A}}(\mathfrak{K}) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{hvzk-1}}_{\mathcal{A}}(\mathfrak{K}) = 1\right]| \leq \varepsilon$, where the experiments $\mathsf{Exp}^{\text{hvzk}-0}_{\mathcal{A}}$ and $\mathsf{Exp}^{\text{hvzk}-1}_{\mathcal{A}}$ are defined as follows:

$\mathsf{Exp}^{\text{hvzk}-0}_{\mathcal{A}}(\mathfrak{K})$
$\sigma \xleftarrow{\$} \mathsf{Setup}(1^{\mathfrak{K}})$
$(\mathsf{st}, x, (w, y)) \xleftarrow{\$} \mathcal{A}(\sigma)$
$b \xleftarrow{\$} \langle \mathsf{Prove}((w, y)), \mathsf{Ver}\rangle_{\sigma}(x)$
let tr be the previous transcript
let r be the random tape of Ver
return $\mathcal{A}(\mathsf{st}, \mathsf{tr}, r, b)$

$\mathsf{Exp}^{\text{hvzk}-1}_{\mathcal{A}}(\mathfrak{K})$
$(\sigma, \tau) \xleftarrow{\$} \mathsf{Sim}_1(1^{\mathfrak{K}})$
$(\mathsf{st}, x, (w, y)) \xleftarrow{\$} \mathcal{A}(\sigma)$
$b \xleftarrow{\$} \langle \mathsf{Sim}_2(\tau), \mathsf{Ver}\rangle_{\sigma}(x)$
let tr be the previous transcript
let r be the random tape of Ver
return $\mathcal{A}(\mathsf{st}, \mathsf{tr}, r, b)$

where st is the state of the adversary.

Witness-Indistinguishable Arguments (WIE-ZK). A witness-indistinguishable argument (WIE-ZK) system for a witness relation $\mathcal{R}$ is a tuple $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2), \mathsf{Ext})$, which verifies the same properties as an E-ZK, except the zero-knowledge property is replaced by the following weaker property:

– *Witness-Indistinguishable Zero-Knowledge.* Π is (t, ε)-witness-indistinguishable if, for any adversary $\mathcal{A}$ running in time at most t, $|\Pr\left[\mathsf{Exp}^{\text{wi-0}}_{\mathcal{A}}(\mathfrak{K}) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{wi-1}}_{\mathcal{A}}(\mathfrak{K}) = 1\right]| \leq \varepsilon$, where the experiments $\mathsf{Exp}^{\text{wi}-b}_{\mathcal{A}}$ (with $b \in \{0, 1\}$) are defined as follows:

$\mathsf{Exp}^{\text{wi}-b}_{\mathcal{A}}(\mathfrak{K})$
$\sigma \xleftarrow{\$} \mathsf{Setup}(1^{\mathfrak{K}})$
$(\mathsf{st}, x, (w_0, y_0), (w_1, y_1)) \xleftarrow{\$} \mathcal{A}(\sigma)$
if $\mathcal{R}(x, (w_0, y_0)) = 0$ or $\mathcal{R}(x, (w_1, y_1)) = 0$ **then**
$b' \xleftarrow{\$} \{0, 1\}$
else
$b' \xleftarrow{\$} \langle \mathsf{Prove}((w_b, y_b)), \mathcal{A}(\mathsf{st})\rangle_{\sigma}(x)$
return b'

where st is the state of the adversary.

Simulation-(Partially)-Extractable Zero-Knowledge Arguments (SE-ZK). A simulation-(partially)-extractable zero-knowledge arguments (SE-ZK) system for a witness relation $\mathcal{R}$ is a tuple $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2), \mathsf{Ext})$, which verifies the same properties as an E-ZK and an additional property:

– *Simulation (Partial) Extractability.* Π is (t, ε)-simulation-(partially)-extractable if, for any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ and $\mathcal{A}_2$ are coordinated and run in time at most t, $\Pr\left[\mathsf{Exp}^{\text{sim-ext}}_{\mathcal{A}}(\mathfrak{K}) = 1\right] \leq \varepsilon$, where the experiment $\mathsf{Exp}^{\text{sim-ext}}_{\mathcal{A}}$ is defined as follows:

$\mathsf{Exp}^{\text{sim-ext}}_{\mathcal{A}}(\mathfrak{K})$
$(\sigma, \tau) \xleftarrow{\$} \mathsf{Sim}_1(1^{\mathfrak{K}})$
$(x, \mathsf{tr}, (b, w)) \xleftarrow{\$} (\langle \boxed{\mathsf{Sim}_2(\tau)}, \mathcal{A}_1\rangle, \langle \mathcal{A}_2, \boxed{\mathsf{Ext}}_1\rangle)_{\sigma}$
let Q be the set of transcripts of $\boxed{\mathsf{Sim}_2}$
if $b = 1$ and $\forall y$, $\mathcal{R}(x, (w, y)) = 0$ and $\forall \mathsf{tr}' \in Q$, $\mathsf{tr} \not\bowtie \mathsf{tr}'$ **then**
return 1
else
return 0

True-Simulation-(Partially)-Extractable Zero-Knowledge Arguments (tSE-ZK). A true-simulation-(partially)-extractable zero-knowledge arguments (tSE-ZK) system for a witness relation $\mathcal{R}$ is a tuple $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2), \mathsf{Ext})$, which verifies the same properties as an E-ZK and an additional property:

– *True Simulation (Partial) Extractability.* The argument Π is (t, ε)-true-simulation-(partially)-extractable if, for any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ and $\mathcal{A}_2$ are coordinated and run in time at most t, $\Pr\left[\mathsf{Exp}^{\text{tsim-ext}}_{\mathcal{A}}(\mathfrak{K}) = 1\right] \leq \varepsilon$, where the experiment $\mathsf{Exp}^{\text{tsim-ext}}_{\mathcal{A}}$ is defined as follows:

$\mathsf{Exp}^{\text{tsim-ext}}_{\mathcal{A}}(\mathfrak{K})$
$(\sigma, \tau) \xleftarrow{\$} \mathsf{Sim}_1(1^{\mathfrak{K}})$
$(x, \mathsf{tr}, (b, w)) \xleftarrow{\$} (\langle \boxed{\mathsf{Sim}'(\tau)}, \mathcal{A}_1\rangle, \langle \mathcal{A}_2, \boxed{\mathsf{Ext}}_1\rangle)_\sigma$
let Q be the set of transcripts of $\boxed{\mathsf{Sim}'}$
if $b = 1$, $\forall y$, $\mathcal{R}(x, (w, y)) = 0$ and $\forall \mathsf{tr}' \in Q$, $\mathsf{tr} \not\bowtie \mathsf{tr}'$ **then**
  return 1
else
  return 0

where $\mathsf{Sim}'(\tau)$ takes as common input a statement x and as private input a witness (w, y), runs $\mathsf{Sim}_2(\tau)$ with common input x if $\mathcal{R}(x, (w, y)) = 1$ and aborts otherwise.

The only difference between true simulation extractability and simulation extractability is that the latter holds when the adversary has access to proofs of any statements (including false statements) whereas the former holds only when the adversary has access to proofs of true statements.

B.2 Σ-Protocols and Ω-Protocols

Let us recall the definition of Σ-protocols and Ω-protocols (as defined in [GMY06]).

Σ-Protocol. Let us consider a language $\mathcal{L}$ defined by a relation $\mathcal{R}$ as in Section 1. For Σ-protocols, we suppose that there is no extractable part of the witness, *i.e.*, $w = \perp$. A Σ-protocol is a three-flow honest-verifier zero-knowledge proof, where the prover first sends a message com called commitments, then the verifier sends back a message $\mathfrak{C}$ called challenge, and finally the prover answers with a message resp called response. In this article, the challenge $\mathfrak{C}$ is always a random element of $\mathbb{Z}_p$.

In addition to completeness, soundness and honest-verifier zero-knowledge, Σ-protocol has to verify the following additional property: weak special soundness, which says that if $(\mathsf{com}, \mathfrak{C}, \mathsf{resp})$ and $(\mathsf{com}, \mathfrak{C}', \mathsf{resp}')$, with $\mathfrak{C} \neq \mathfrak{C}'$, are two accepting transcripts for some word x, then necessarily, $x \in \mathcal{L}$. It is easy to see that the weak special soundness directly implies the soundness.

Ω-Protocol. Ω-protocols are extensions of Σ-protocols which are in addition partially extractable. Therefore an Ω-protocol is a HVE-ZK. We remark that what we call Ω-protocols in our article are called f-extracting Ω-protocols in [GMY06], with f defined as follows: $f((w, y)) = w$.

In our article, Ω-protocols are always constructed as follows: the prover encrypts the partial witness w under an encryption key ek in the CRS crs and proves, using a Σ-protocol, that he has done so.

C Details on Zero-Knowledge Arguments from SPHFs and TSPHFs

C.1 SPHFs for Multi-Exponentiation Equations

In this section, we show there exists a KV-SPHF for languages of ElGamal ciphertexts (with or without reuse of randomness) of elements $(X_1, \ldots, X_n) \in \mathbb{G}^n$ verifying:

$$\exists (y_1, \ldots, y_m) \in \mathbb{Z}_p^m,\ \forall k \in \{1, \ldots, t\},\ \prod_{i=1}^{n} X_i^{a_{k,i}} = \prod_{j=1}^{m} A_{k,j}^{y_j} \cdot B_k,$$

where the $a_{k,i}$'s and $A_{k,i}$'s are constants. This yields an HVE-ZK for the multi-exponentiation equations language described in Section 6.2. In this section i, j and k will always range respectively from 1 to n, from 1 to m and from 1 to t.

Without Reuse of Randomness. Let us write $C = (C_1, \ldots, C_n)$ the ciphertext of the plaintext $(X_1, \ldots, X_n)$ with $C_i = (u_i = g^{r_i}, e_i = h^{r_i} \cdot X_i)$. We build the KV-SPHF as follows:

$$\Gamma = \left(\begin{array}{cc|cc} g & & 1 & \\ & \ddots & & \ddots \\ 1 & & g & \\ & & & h \\ \hline & 1 & A_{1,1} & \cdots & A_{t,1} \\ & & \vdots & & \vdots \\ & & A_{1,m} & \cdots & A_{t,m} \end{array}\right) \qquad \begin{aligned} \Theta_{\mathsf{aux}}(\boldsymbol{C}) &= \left(\left(\textstyle\prod_{i=1}^{n} u_i^{a_{k,i}}\right)_k, \left(\textstyle\prod_{i=1}^{n} e_i^{a_{k,i}} / B_k\right)_k\right) \\ \boldsymbol{\lambda} &= \left(\left(\textstyle\sum_{i=1}^{n} a_{k,i} r_i\right)_i, (y_j)_j\right) \\ \boldsymbol{\lambda} \cdot \Gamma &= \left(\left(\textstyle\prod_{i=1}^{n} g^{a_{k,i} r_i}\right)_k, \left(\textstyle\prod_{i=1}^{n} h^{a_{k,i} r_i} \cdot \prod_{j=1}^{m} A_{k,j}^{y_j}\right)_k\right). \end{aligned}$$

With Reuse of Randomness. Let us write $C = (u = g^r, e_1 = h_1^r \cdot X_1, \ldots, e_n = h_n^r \cdot X_n)$. We build the KV-SPHF as follows:

$$\Gamma = \left(\begin{array}{c|ccc} g & \prod_{i=1}^{n} h_i^{a_{1,i}} & \cdots & \prod_{i=1}^{n} h_i^{a_{t,i}} \\ \hline 1 & A_{1,1} & \cdots & A_{t,1} \\ \vdots & \vdots & & \vdots \\ 1 & A_{1,m} & \cdots & A_{t,m} \end{array}\right) \qquad \begin{aligned} \Theta_{\mathsf{aux}}(\boldsymbol{C}) &= \left(u, \left(\textstyle\prod_{i=1}^{n} e_i^{a_{k,i}} / B_k\right)_k\right) \\ \boldsymbol{\lambda} &= (r, (y_j)_j) \\ \boldsymbol{\lambda} \cdot \Gamma &= \left(g^r, \left(\textstyle\prod_{i=1}^{n} h_i^{a_{k,i} r} \cdot \prod_{j=1}^{m} A_{k,j}^{y_j}\right)_k\right). \end{aligned}$$

C.2 Ω-Protocols for Multi-Exponentiation Equations

In this section, we present Ω-protocols for the multi-exponentiation equations language described in Section 6.2:

$$\forall (X_1, \ldots, X_n) \in \mathbb{G}^n,\ \exists (y_1, \ldots, y_m) \in \mathbb{Z}_p^m,\ \forall k \in \{1, \ldots, t\},\ \prod_{i=1}^{n} X_i^{a_{k,i}} = \prod_{j=1}^{m} A_{k,j}^{y_j} \cdot B_k.$$

The first one uses ElGamal ciphertexts without reuse of randomness and the second one uses ElGamal ciphertexts with reuse of randomness. In this section i, j and k will always range respectively from 1 to n, from 1 to m and from 1 to t. The two constructions use classical methods from [Sch91].

Without Reuse of Randomness.

- commitment: $\mathsf{com} = (C, (u'_k)_k, (e'_k)_k)$ with $C = (C_1, \ldots, C_n)$ an ElGamal ciphertext of the plaintext $(X_1, \ldots, X_n)$ ($C_i = (u_i = g^{r_i}, e_i = h^{r_i} \cdot X_i)$), $u'_k = g^{\sum_{i=1}^n a_{k,i} r'_i}$ and $e'_k = h^{\sum_{i=1}^n a_{k,i} r'_i} \cdot \prod_{j=1}^m A_{k,j}^{y'_j}$ with $r_i, r'_i, y'_j \xleftarrow{\$} \mathbb{Z}_p$;
- challenge: $\mathfrak{C} \xleftarrow{\$} \mathbb{Z}_p$;
- response: $\mathsf{resp} = ((r''_k)_k, (y''_j)_j)$ with $r''_k = \sum_{i=1}^n a_{k,i}(r'_i + \mathfrak{C} r_i)$ and $y''_j = y'_j + \mathfrak{C} y_j$;
- verification:

$$\prod_{i=1}^n u_i^{\mathfrak{C} a_{k,i}} \cdot u'_k = g^{r''_k} \qquad\qquad \prod_{i=1}^n e_i^{\mathfrak{C} a_{k,i}} \cdot e'_k = h^{r''_k} \cdot \prod_{j=1}^m A_{k,j}^{y''_j} \cdot B_k^{\mathfrak{C}}.$$

With Reuse of Randomness.

- commitment: $\mathsf{com} = (C, u', (e'_k)_k)$ with $C = (u = g^r, e_1 = h_1^r \cdot X_1, \ldots, e_n = h_n^r \cdot X_n)$ an ElGamal ciphertext of the plaintext $(X_1, \ldots, X_n)$, $u' = g^{r'}$, $e'_k = \prod_{i=1}^n h_i^{a_{k,i} r'} \cdot \prod_{j=1}^m A_{k,j}^{y'_j}$ with $r, r', y'_j \xleftarrow{\$} \mathbb{Z}_p$;
- challenge: $\mathfrak{C} \xleftarrow{\$} \mathbb{Z}_p$;
- response: $\mathsf{resp} = (r'', (y''_j)_j)$ with $r'' = r' + \mathfrak{C} r$ and $y''_j = y'_j + \mathfrak{C} y_j$;
- verification:

$$u^{\mathfrak{C}} \cdot u' = g^{r''} \qquad\qquad \prod_{i=1}^n e_i^{\mathfrak{C} a_{k,i}} \cdot e'_k = \prod_{i=1}^n h_i^{a_{k,i} r''} \cdot \prod_{j=1}^m A_{k,j}^{y''_j} \cdot B_k^{\mathfrak{C}}.$$

C.3 Instantiations of tSE-ZK

In this section, we extend the multi-exponentiation KV-SPHF of Appendix C.1 from ElGamal ciphertexts with reuse of randomness to Cramer-Shoup ciphertexts with reuse of randomness[4]. This construction is done as explained in [BBC+13], with some additional optimizations. This yields an efficient tSE-ZK. We use the same notation as in the original construction, except for ciphertexts.

Let us write $C = (u_1 = g_1^r, u_2 = g_2^r, e_1 = h_1^r \cdot X_1, \ldots, e_n = h_n^r \cdot X_n, v = (cd^\xi)^r)$, for which we know $(y_1, \ldots, y_m) \in \mathbb{Z}_p^m$ such that

$$\forall k \in \{1, \ldots, t\}, \prod_{i=1}^n X_i^{a_{k,i}} = \prod_{j=1}^m A_{k,j}^{y_j} \cdot B_k.$$

We build the KV-SPHF as follows:

$$\Gamma = \left(\begin{array}{c|c|c|c|c} g_1 & 1 & g_2 & \prod_{i=1}^n h_i^{a_{1,i}} \cdots \prod_{i=1}^n h_i^{a_{t,i}} & c \\ \hline 1 & g_1 & 1 & 1 \cdots 1 & d \\ \hline 1 & 1 & 1 & A_{1,1} \cdots A_{t,1} & 1 \\ \vdots & \vdots & \vdots & \vdots \quad \vdots & \vdots \\ 1 & 1 & 1 & A_{1,m} \cdots A_{t,m} & 1 \end{array}\right)$$

$$\Theta_{\mathsf{aux}}(\boldsymbol{C}) = \left(u_1, u_1^\xi, u_2, \left(\textstyle\prod_{i=1}^n e_i^{a_{k,i}} / B_k\right)_k, v\right)$$
$$\boldsymbol{\lambda} = (r, r\xi, (y_j)_j)$$
$$\boldsymbol{\lambda} \cdot \Gamma = \left(g_1^r, g_1^{r\xi}, g_2^r, \left(\textstyle\prod_{i=1}^n h_i^{a_{k,i} r} \cdot \prod_{j=1}^n A_{k,j}^{y_j}\right)_k, (cd^\xi)^r\right).$$

[4] The case without reuse of randomness is easier and cannot be easily optimized contrary to the case with reuse of randomnness.

D Security Proofs

D.1 Proof for Our Generic One-Round UC-Secure PAKE

In this section, we prove the our generic one-round PAKE described in Section 5.1 is secure in the UC framework against static corruptions, with a common reference string for any TSPHF on the language of a valid ciphertext on a message m under a IND-CCA-secure labeled encryption scheme. More precisely, for any environment trying to distinguish a real execution from an ideal one with the simulator given below, within time t, one can show that its advantage ε is bounded by

$$q \times \left((q+1) \cdot \mathsf{Adv}^{\mathsf{smooth}}(t) + 3 \cdot \mathsf{Adv}^{\mathsf{ind\text{-}cca}}(t, q_S) + \mathsf{Adv}^{\mathrm{sound}}(t) \right),$$

where q_S is the number of sessions and q the number of activated players, and $\mathsf{Adv}^{\mathsf{smooth}}(t)$ is the best advantage one can get in the smoothness security game within time t, while $\mathsf{Adv}^{\mathsf{ind}}(t, q)$ is the best advantage one can get in the IND-CCA security game within time t, and with at most q decryption queries, and $\mathsf{Adv}^{\mathrm{sound}}(t)$ is the best advantage one can get in the soundness security game within time t.

The proof follows that of [KV11], but it is a bit more involved for two reasons:

- we do not assume a prior agreement of the session ID. Such an assumption would limit the interest of a one-round protocol since this prior agreement would need an additional round of nonces;
- our TSPHF does not guarantee the smoothness when the trapdoor τ' is known, and then, we have to modify the order of the games to use this trapdoor at the very end only.

We insist we are in the static-corruption model, the adversary can only corrupt players before the execution of the protocol. We thus assume players to be honest or not at the beginning, and they cannot be corrupted afterwards. However, this does not prevent the adversary from modifying flows coming from the players: following [CHK+05], we say that a flow is oracle-generated if it was sent by an honest player and not altered. Otherwise, we say it is non-oracle-generated.

Description of the Simulator $\mathcal{S}$ The simulator $\mathcal{S}$ first generates the CRS, with an encryption key ek, while knowing the decryption key dk, and the parameters crs for the TSPHF, with the trapdoor τ'. Note that ek might be included in crs. We furthermore assume 1 can never be a valid password. Otherwise, we can replace 1 by any other value.

Receiving a $(\mathsf{NewSession} : \mathsf{sid}, P_i, P_j)$ *from the ideal functionality.* This indicates that P_i should initiate the protocol with P_j. $\mathcal{S}$ generates hashing and projection keys $(\mathsf{hk}, \mathsf{hp})$ for P_i, as well as an encryption C of 1, with the label $\ell = (P_i, P_j, \mathsf{hp})$. It sends the message (hp, C) to $\mathcal{A}$.

Receiving a message (hp', C') *from* $\mathcal{A}$. Let us denote P_i the uncorrupted player to whom $\mathcal{A}$ sent this message.

1. if (hp', C') has been oracle-generated, and more precisely by $\mathcal{S}$ on behalf of P_j, then $\mathcal{S}$ sends $(\mathsf{NewKey} : \mathsf{sid}, P_i, \perp)$ to the functionality. This makes P_i to receive a random session key if it terminates before P_j or if the passwords differ, or otherwise the same session key as the one sent to P_j.
2. Otherwise, $\mathcal{S}$ uses dk to extract pw′ from C'. Then $\mathcal{S}$ asks for $(\mathsf{TestPW} : \mathsf{sid}, P_i, \mathsf{pw}')$ to the functionality and gets back either "correct guess" or "wrong guess". If the reply is "correct guess", then $\mathsf{full\text{-}aux} = (\mathsf{crs}, \mathsf{pw}')$ and $\mathcal{S}$ uses the global trapdoor τ' to compute $H = \mathsf{THash}(\mathsf{hp}', \mathsf{full\text{-}aux}, C, \tau')$ and can compute $H' = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C')$, and $\mathsf{sk} = H \cdot H'$. Finally, $\mathcal{S}$ sends $(\mathsf{NewKey} : \mathsf{sid}, P_i, \mathsf{sk})$ to the functionality, which makes P_i to receive sk. If the reply is "wrong guess", then $\mathcal{S}$ sends $(\mathsf{NewKey} : \mathsf{sid}, P_i, \perp)$ to the functionality, which makes P_i to receive a random session key.

Sequence of Games We now provide the complete proof by a sequence of games, that starts from the real game and ends with the above simulation. Each steps are shown indistinguishable under specific assumptions.

Game $\mathbf{G}_0$: This is the real game, where the CRS is correctly generated and every flow from honest players are generated correctly by the simulator which knows the inputs sent by the environment to the players:
- Upon receiving a $(\mathsf{NewSession} : \mathsf{sid}, P_i, P_j)$ from the ideal functionality, $\mathcal{S}$ generates hashing and projection keys $(\mathsf{hk}, \mathsf{hp})$ for P_i, and encrypts pw into C with random coins r, under the label $\ell = (P_i, P_j, \mathsf{hp})$. It sends (hp, C) to $\mathcal{A}$;
- Upon receiving (hp', C') from $\mathcal{A}$ on behalf of P_j, $\mathcal{S}$ computes $H = \mathsf{ProjHash}(\mathsf{hp}', \mathsf{full\text{-}aux}, C, r)$ and $H' = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C')$, where $\mathsf{full\text{-}aux} = (\mathsf{crs}, \mathsf{crs}', \mathsf{pw})$, with the appropriate labels, and provides $\mathsf{sk} = H \cdot H'$ to P_i.

Game $\mathbf{G}_1$: In this game, the CRS is generated by the simulator that knows the decryption key dk. This does not change anything, and thus keeps the game perfectly indistinguishable.

Game $\mathbf{G}_2$: Upon receiving (hp', C') from $\mathcal{A}$ on behalf of P_j, either oracle-generated with password pw' and hashing key hk', or non-oracle-generated, with C' that decrypts to pw' (possibly $\perp$), $\mathcal{S}$ compares pw with pw'. If they are different, $\mathcal{S}$ chooses H' at random instead of $H' = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C')$.

Using the hybrid argument with the smoothness security game (where the decryption key dk is known and thus allows to test the language-membership), one shows this game is indistinguishable from the former one since the Hash evaluation with a ciphertext not in the language is used in this case only, all the other evaluations are for ciphertexts in the language: the distance is bounded by $q \cdot \varepsilon^{\mathsf{smooth}}$, where q is the number of activated players and thus less than twice the number q_S of sessions, and $\varepsilon^{\mathsf{smooth}}$ is the best advantage against the smoothness one can get within the execution time of the full game (the initial running time of the environment plus the simulations that essentially make additional decryption evaluations).

Game $\mathbf{G}_3$: Upon receiving (hp', C') from $\mathcal{A}$ on behalf of P_j, either oracle-generated with password pw' and hashing key hk', or non-oracle-generated, with C' that decrypts to pw' (possibly $\perp$), $\mathcal{S}$ compares pw with pw'. If they are different, $\mathcal{S}$ does not compute anymore H and H', but chooses sk at random. This game is perfectly indistinguishable from the former one, since sk was the product of H with a random H'.

Game $\mathbf{G}_4$: Upon receiving (hp', C') from $\mathcal{A}$ on behalf of P_j, either oracle-generated by $\mathcal{S}$ on behalf of a player with password pw' and hashing key hk', or non-oracle-generated, with C' that decrypts to pw' (possibly $\perp$) and unknown hashing key, we thus note $\mathsf{hk}' = \perp$, $\mathcal{S}$ compares pw with pw'. If they are the same, $\mathcal{S}$ computes $H = \mathsf{ProjHashL}(\mathsf{hp}', \mathsf{hk}', \mathsf{full\text{-}aux}, C, r)$, instead of $H = \mathsf{ProjHash}(\mathsf{hp}', \mathsf{full\text{-}aux}, C, r)$, and $H' = \mathsf{HashL}(\mathsf{hk}, \mathsf{full\text{-}aux}, C')$ instead of $H' = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C')$, where HashL and $\mathsf{ProjHashL}$ work as follows with a list Λ_H initially set to an empty:
- for a query $\mathsf{HashL}(\mathsf{hk}, \mathsf{full\text{-}aux}, C')$, if there is h such that $(\mathsf{hk}, \mathsf{full\text{-}aux}, C'; h) \in \Lambda_H$, then h is returned, otherwise one computes $h = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C')$, then $(\mathsf{hk}, \mathsf{full\text{-}aux}, C'; h)$ is appended to Λ_H, and h is returned;
- for a query $\mathsf{ProjHashL}(\mathsf{hp}, \mathsf{hk}, \mathsf{full\text{-}aux}, C', r')$, if there is h such that $(\mathsf{hk}, \mathsf{full\text{-}aux}, C'; h) \in \Lambda_H$, then h is returned, otherwise one computes $h = \mathsf{ProjHash}(\mathsf{hp}, \mathsf{full\text{-}aux}, C', r')$, then $(\mathsf{hk}, \mathsf{full\text{-}aux}, C'; h)$ is appended to Λ_H, and h is returned.

Because of the hash correctness, this game is perfectly indistinguishable from the former one since some evaluations of $\mathsf{ProjHash}(\mathsf{hp}', \mathsf{full\text{-}aux}, C, r)$ are replaced by $\mathsf{Hash}(\mathsf{hk}', \mathsf{full\text{-}aux}, C)$ (or vice-versa, when already computed), but only for associated hashing and projection keys on valid ciphertexts.

Game $\mathbf{G}_5$: Upon receiving (hp', C') from $\mathcal{A}$ on behalf of P_j, but oracle-generated by $\mathcal{S}$ on behalf of a player with password pw' and hashing key hk', $\mathcal{S}$ compares pw with pw'. If they are the same, $\mathcal{S}$ still computes $H = \mathsf{ProjHashL}(\mathsf{hp}', \mathsf{hk}', \mathsf{full\text{-}aux}, C, r)$ but $H' = \mathsf{HashL}'(\mathsf{hk}, \mathsf{full\text{-}aux}, C')$ instead of using HashL, where one chooses h at random when not yet defined.

In order to prove the indistinguishability of this game from the former one, a series of hybrid games is required: In $\mathbf{G}(i)$, upon receiving the j-th oracle-generated flow (hp', C') from $\mathcal{A}$ with identical passwords,

if $j \leq i$, one uses HashL', otherwise one uses HashL. One can note that $\mathbf{G}(0)$ is $\mathbf{G}_4$, and $\mathbf{G}(q)$ is $\mathbf{G}_5$, where q is the number of activated players, and thus less than twice the number q_S of sessions.
For any i, let us name $\mathbf{G}(i)$ as $\mathbf{G}_A$, where we also name the i-th oracle-generated flow (hp', C') from $\mathcal{A}$ with identical passwords the *critical flow* (note that the same pair can be sent multiple times, but we focus on a flow, defined by the generator and the receiver). We additionally name the generator of this flow P_a (where a is the index of the $\mathsf{NewSession}$-query for this player), and the receiver of this flow P_b (where b is the index of the $\mathsf{NewSession}$-query for this player).
In this game, we guess the index a. We thus expect the a-th $\mathsf{NewSession}$-query to generate the pair (hp', C') sent in the critical flow. In case of bad guess, one aborts and restarts. We succeed with probability greater than $1/q$.
We stress that rewinding is not possible in the UC framework, but here, we just want to show that no environment can distinguish the games. If an environment can, by aborting some executions (at random, which is the case here since the guess is independent from the execution), we just have a distinguisher that either makes q more time, on average, to answer, with the same bias; or within the same time answers with a bias reduced by a factor q (by simply answering at random in case of abort).

- In $\mathbf{G}_B$, when P_a (the generator) receives its flow, one uses $\mathsf{ProjHashL}'$, instead of $\mathsf{ProjHashL}$, where in the evaluation of $\mathsf{ProjHashL}'(\mathsf{hp}', \mathsf{hk}', \mathsf{full\text{-}aux}, C, r)$, h is computed as $\mathsf{Hash}(\mathsf{hk}', \mathsf{full\text{-}aux}, C)$ instead of $\mathsf{ProjHash}(\mathsf{hp}', \mathsf{full\text{-}aux}, C, r)$. Actually, $\mathsf{ProjHashL}'(\mathsf{hp}', \mathsf{hk}', \mathsf{full\text{-}aux}, C, r) = \mathsf{HashL}(\mathsf{hk}', \mathsf{full\text{-}aux}, C)$.
  Under the hash correctness, this game is perfectly indistinguishable from $\mathbf{G}_A$, since one evaluation of $\mathsf{ProjHash}(\mathsf{hp}', \mathsf{full\text{-}aux}, C, r)$ might be replaced by $\mathsf{Hash}(\mathsf{hk}', \mathsf{full\text{-}aux}, C)$ but only for associated hashing and projection keys on a valid ciphertext: note that here, C is the ciphertext in the critical flow.
- In $\mathbf{G}_C$, for the a-th $\mathsf{NewSession}$-query, one encrypts 1 instead of pw.
  Under the $\mathsf{IND\text{-}CCA}$-security of the encryption scheme, one can show this game is indistinguishable from $\mathbf{G}_B$. We note that we can apply $\mathsf{IND\text{-}CCA}$ security game on the encryption that generates C' from the critical flow: one first remarks that the random coins r of this ciphertext are not needed in $\mathsf{ProjHashL}'$ for P_a, and so the ciphertext can be generated by the encryption oracle. Furthermore, if $\mathcal{A}$ tries to replay the ciphertext, it can either be with the same projection key, and then no decryption is needed since this is an oracle-generated flow, or with a different projection key and thus under a different label, which allows the decryption query. The distance is thus bounded by $\varepsilon^{\mathsf{ind}}$, the advantage an adversary can have within the same time as our distinguisher, against the $\mathsf{IND\text{-}CCA}$ security of the encryption scheme after at most q_S decryption queries, where q_S is the number of sessions.
- In $\mathbf{G}_D$, when P_b receives its flow (the critical flow), one uses HashL', instead of HashL.
  With the smoothness security game on the specific keys generated by $\mathcal{S}$ on behalf of P_b, we can show that the distance is bounded by $\varepsilon^{\mathsf{smooth}}$. In addition, we know that this is always a word non in the language.
  But again, to apply the smoothness security game on the good keys, one has to guess b and to either restart in case of bad guess or output a random answer for the distinction between the real or random hash value.
- In $\mathbf{G}_E$, for the a-th $\mathsf{NewSession}$-query, one encrypts back pw instead of 1.
  Exactly as above, under the $\mathsf{IND\text{-}CCA}$-security of the encryption scheme, one can show this game is indistinguishable from the $\mathbf{G}_D$ and the distance is thus bounded by $\varepsilon^{\mathsf{ind}}$.
- In $\mathbf{G}_F$, when P_a (the generator) receives its flow, one uses back $\mathsf{ProjHashL}$, instead of $\mathsf{ProjHashL}'$.
  Under the hash correctness, this game is perfectly indistinguishable from $\mathbf{G}_E$, since one evaluation of $\mathsf{Hash}(\mathsf{hk}', \mathsf{full\text{-}aux}, C)$ might be replaced by $\mathsf{ProjHash}(\mathsf{hp}', \mathsf{full\text{-}aux}, C, r)$ but only for associated hashing and projection keys on a valid ciphertext: note that here, C is the ciphertext in the critical flow.

If we assume we answer a random bit when aborting, the distance between $\mathbf{G}_F$ and $\mathbf{G}_A$ is $q \cdot \varepsilon^{\mathsf{smooth}} + 2 \cdot \varepsilon^{\mathsf{ind}}$.

Game $\mathbf{G}_6$: Upon receiving (hp', C') from $\mathcal{A}$ on behalf of P_j, but oracle-generated by $\mathcal{S}$ on behalf of a player with password pw' and hashing key hk', $\mathcal{S}$ compares pw with pw'. If they are the same, $\mathcal{S}$ chooses sk at

random, and will provide the same key at the partner, when it will receive (hp, C) from $\mathcal{A}$ on behalf of P_i.

This game is perfectly indistinguishable from the former one, since the first player that computes sk makes a product of H' with a random H. And because of the list Λ_H in $\mathsf{ProjHashL}$ and HashL', the partner will compute the same H and H', and thus the same sk.

Game $\mathbf{G}_7$: In this game, the CRS is fully generated by the simulator that then knows both the decryption key dk and the general trapdoor τ'.

This keeps this game perfectly indistinguishable from the former one.

Game $\mathbf{G}_8$: Upon receiving (hp', C') from $\mathcal{A}$ on behalf of P_j, non-oracle-generated, with C' that decrypts to pw' (possibly $\perp$), $\mathcal{S}$ compares pw with pw'. If they are the same, $\mathcal{S}$ computes $H = \mathsf{THash}(\mathsf{hp}', \mathsf{full\text{-}aux}, C, \tau')$ instead of $H = \mathsf{ProjHash}(\mathsf{hp}', \mathsf{full\text{-}aux}, C, r)$, but still $H' = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C')$.

Because of the soundness, this game is indistinguishable from the former one.

Game $\mathbf{G}_9$: Upon receiving a $(\mathsf{NewSession} : \mathsf{sid}, P_i, P_j)$ from the ideal functionality, $\mathcal{S}$ generates hashing and projection keys $(\mathsf{hk}, \mathsf{hp})$ for P_i, and encrypts 1 (instead of pw) into C, under the label $\ell = (P_i, P_j, \mathsf{hp})$. It sends (hp, C) to $\mathcal{A}$.

Under the $\mathsf{IND\text{-}CCA}$-security of the encryption scheme, one can show this game is indistinguishable from the former one. To apply the $\mathsf{IND\text{-}CCA}$ security game, one first remarks that the random coins r are not needed anymore to compute H in any case, and so C can be generated by the encryption oracle. Furthermore, if $\mathcal{A}$ tries to replay a ciphertext C generated by the challenger on behalf of P_j with password pw', it can either be with the same projection key, and then no decryption is needed since this is an oracle-generated flow, or with a different projection key and thus under a different label, which allows the decryption query. The distance is thus bounded by $q \cdot \varepsilon^{\mathsf{ind}}$.

The current game is thus the following one:

- Upon receiving a $(\mathsf{NewSession} : \mathsf{sid}, P_i, P_j)$ from the ideal functionality, $\mathcal{S}$ generates hashing and projection keys $(\mathsf{hk}, \mathsf{hp})$ for P_i, and encrypts 1 into C. It sends (hp, C) to $\mathcal{A}$;
- Upon receiving (hp', C') from $\mathcal{A}$ on behalf of P_j,
  - if (hp', C') is oracle-generated with password pw' and hashing key hk', $\mathcal{S}$ compares pw with pw'.
    - If they are the same, $\mathcal{S}$ chooses sk at random, and will give the same key at the partner;
    - If they are different, $\mathcal{S}$ chooses sk at random.
  - if (hp', C') is non-oracle-generated, with C' that decrypts to pw' (possibly $\perp$), $\mathcal{S}$ compares pw with pw'.
    - If they are the same, the simulator $\mathcal{S}$ computes $H = \mathsf{THash}(\mathsf{hp}', \mathsf{full\text{-}aux}, C, \tau')$, and $H' = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C')$, where $\mathsf{full\text{-}aux} = (\mathsf{crs}, \mathsf{crs}', \mathsf{pw})$, and provides $\mathsf{sk} = H \cdot H'$ to P_i;
    - If they are different, $\mathcal{S}$ chooses sk at random.

Game $\mathbf{G}_{10}$: In this game, $\mathcal{S}$ now uses the ideal functionality as explained in the description of the simulator in Appendix D.1. It is easy to see that this game is perfectly indistinguishable from the former one.

D.2 Proof for the E-ZK and tSE-ZK Construction

In this section, we prove the true-simulation extractability of the two $\mathsf{tSE\text{-}ZK}$ constructions from TSPHFs in Section 7.1. Using the last experiments of both proofs, we can derive a proof for the soundness and the extractability of the $\mathsf{E\text{-}ZK}$ constructions.

Two-Flow Construction. Here is a sequence of indistinguishable experiments proving the true-simulation extractability of the two-flow construction.

Experiment $\mathbf{E}_0$: This first experiment is the experiment of the definition of true-simulation extractability (Section B.1). In particular, on input $(x, (w, y))$, Sim' checks whether $\mathcal{R}(x, (w, y)) = 1$, and if it is the

case, it simulates the argument using Sim_2, which encrypts an arbitrary value in C instead of w and which uses THash and the trapdoor τ' of the TSPHF to compute its hash value. In parallel, Ext simulates a verifier against the adversary $\mathcal{A}_2$ playing the role of a prover for some word x, and extracts a partial witness w (or $\perp$ if the decryption fails) for this word x, by decrypting the ciphertext of the adversary. The adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ wins if it manages to make Ext accept and extract an invalid w (*i.e.*, such that $\forall y, \mathcal{R}(x, (w, y)) = 0$) without simply "copying" a transcript of a simulated proof.

Experiment $\mathbf{E}_1$: Let $\mathsf{tr} = (x, (\mathsf{hp}), (C, H))$ be the transcript between $\mathcal{A}_1$ and Ext. In this experiment, the adversary also looses (the experiment returns 0) when the pair $(\ell = (x, \mathsf{hp}), C)$ has been used by one instance of the simulator Sim'. This does not change anything. Indeed, let us suppose (ℓ, C) has been used by one instance of Sim', and let us write $\mathsf{tr}' = (x, (\mathsf{hp}), (C, H'))$ the transcript of this instance. Then either $H' = H$ and $\mathsf{tr} \bowtie \mathsf{tr}'$, and so the adversary also looses; either $H' \neq H$ and so Ext would have rejected the proof of $\mathcal{A}_1$, because only one hash value is valid, and H' is the valid hash value (since it has been computed using THash, which is the same as computed using Hash thanks to the trapdoor correctness and the fact hp has been honestly generated).

Experiment $\mathbf{E}_2$: In this experiment, we change the simulator Sim' to correctly encrypt the partial witnesses w instead of encrypting an arbitrary value. This experiment is indistinguishable from the previous one, because of the encryption scheme is $\mathsf{IND\text{-}CCA}$.

Experiment $\mathbf{E}_3$: In this experiment, we change the simulator Sim' to use $\mathsf{ProjHash}$ instead of THash (possible since it knows the witness (w, y) and the encryption is done correctly). This experiment is indistinguishable from the previous one, thanks to the soundness of the TSPHF.

Experiment $\mathbf{E}_4$: In this experiment, we change Ext to check whether the extracted w is a valid partial witness for x. This is possible using τ. If it is not the case, instead of comparing the hash value H returned by the adversary $\mathcal{A}_2$ with $H' \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, (\ell, C))$ and accepting the proof if and only if $H = H'$, we compare it with $H' \xleftarrow{\$} \Pi$. This experiment is indistinguishable from the previous one, thanks to the computational smoothness of the TSPHF.

The adversary cannot win this last experiment with non-negligible because H' is random and independent from H.

Three-Flow Construction. The same proof can be done, except for the justification that the two first games are indistinguishable. For this justification, we just need to remark that, since the one-time signature is unforgeable, the adversary $\mathcal{A}_2$ cannot output a ciphertext C and a label $\ell = \mathsf{vk}'$ already generated by some instance Sim', without copying the whole transcript tr' or forging a signature under vk.

E Generic Framework for SPHFs and TSPHFs

In this appendix, we first recall the generic framework introduced in [BBC+13], and sketched in Section 4.2. Then, we extend our construction of TSPHFs of Section 4.3 to the full generic framework, and prove the smoothness of the resulting TSPHFs.

This appendix is very formal and technical. We strongly recommend the reader to first read Sections 4.2 and 4.3, or [BBC+13] (even better) where we give the intuition.

E.1 Recall of the Generic Framework for SPHFs of [BBC+13]

Graded Rings. Before introducing the generic framework, we first need to recall the notion of graded ring introduced in [BBC+13]. Graded rings are a generalization of bilinear settings and a practical way to manipulate elements of various groups involved with pairings, and more generally, with multi-linear maps.

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

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

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

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

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

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

Cyclic Groups and Pairing-Friendly Settings. In the sequel, we consider graded rings over $R = \mathbb{Z}_p$ only, because we will use the vectorial space structure over $\mathbb{Z}_p$ in the proof of the smoothness of our generic construction of SPHF. This means we cannot directly deal with constructions in [GGH12] yet. Nevertheless, graded rings enable to easily deal with cyclic groups $\mathbb{G}$ of prime order p, and bilinear groups.

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 $g^x \in \mathbb{G}$.

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 $e(g, g)^x \in \mathbb{G}_T$.

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 $e(g_1, g_2)^x \in \mathbb{G}_T$.

Projections. We also consider the two following projections: $\mathfrak{I} : \mathfrak{G} \to \Lambda$ and $\mathfrak{L} : \mathfrak{G} \to R$, which on input $u = [\boldsymbol{v}, x]$ output respectively the index $\boldsymbol{v}$ of u and the x part of u, which can be seen as the discrete logarithm of u (hence the notation $\mathfrak{L}$). We remark that for every $u_1, u_2 \in \mathfrak{G}$, $\mathfrak{L}(u_1 + u_2) = \mathfrak{L}(u_1) + \mathfrak{L}(u_2)$ and, if $u_1 \cdot u_2 \neq \perp$, $\mathfrak{L}(u_1 \cdot u_2) = \mathfrak{L}(u_1) \cdot \mathfrak{L}(u_2)$. The projections $\mathfrak{I}$ and $\mathfrak{L}$ can be applied component-wise on vectors and matrices.

Generic Framework for SPHF. In this section, we recall the generic framework for SPHFs for languages of ciphertexts in [BBC+13], with slight modifications, because in this article, hp can depend on aux. Our goal is to deal with languages of ciphertexts $\textsc{Lofc}_{\mathsf{full\text{-}aux}}$: we assume that crs is fixed and we write $\mathrm{L}_{\mathsf{aux}} = \textsc{Lofc}_{\mathsf{full\text{-}aux}} \subseteq \mathcal{S}et$ where $\mathsf{full\text{-}aux} = (\mathsf{crs}, \mathsf{aux})$.

Language Representation. For a language $\mathrm{L}_{\mathsf{aux}}$, we assume there exist two positive integers k and n, and two families of functions $\Gamma_{\mathsf{aux}} : \mathcal{S}et \mapsto \mathfrak{G}^{k \times n}$, and $\Theta_{\mathsf{aux}} : \mathcal{S}et \mapsto \mathfrak{G}^{1 \times n}$, such that for any word $C \in \mathcal{S}et$, $(C \in \mathrm{L}_{\mathsf{aux}}) \iff (\exists \boldsymbol{\lambda} \in \mathfrak{G}^{1 \times k}$ such that $\Theta_{\mathsf{aux}}(C) = \boldsymbol{\lambda} \cdot \Gamma(C))$. If Γ_{aux} is a constant function (independent of the word C), this defines a KV-SPHF, otherwise this defines a GL-SPHF. However, in any case, we need the indexes of the components of $\Gamma_{\mathsf{aux}}(C)$ to be independent of C. We furthermore require that a user, who knows a witness w of the membership $C \in \mathrm{L}_{\mathsf{aux}}$, can efficiently compute $\boldsymbol{\lambda}$.

Smooth Projective Hash Function. With the above notations, the hashing key is a random vector $\mathsf{hk} = \boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^\intercal \xleftarrow{\$} \mathbb{Z}_p^n$, while the projection key is, for a word C, $\mathsf{hp} = \boldsymbol{\gamma}(C) = \Gamma(C) \cdot \boldsymbol{\alpha} \in \mathfrak{G}^k$ (if Γ does not depend on C, hp does not depend on C either). Then, the hash value is:

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

The set Π of hash values is exactly $\mathfrak{G}_{\boldsymbol{v}_H}$, the set of graded elements of index $\boldsymbol{v}_H$, the maximal index of the elements of $\Theta_{\mathsf{aux}}(C)$. It is shown in [BBC+13], that the resulting SPHF is a perfectly smooth KV-SPHF if Γ_{aux} is a constant function and is a perfectly smooth GL-SPHF otherwise. In the sequel, we often write $\boldsymbol{\gamma}$ instead of $\boldsymbol{\gamma}(C)$ and Γ or Γ_{aux} instead of $\Gamma_{\mathsf{aux}}(C)$.

E.2 Efficient Construction of TSPHFs under DDH

In this section, we describe more formally the construction of TSPHFs under DDH of Section 4.3, and, in the same time, we generalize it to SPHFs constructed using the above full generic framework.

More precisely, we show how to construct a TSPHF from any SPHF constructed via the previous framework, under the three following conditions:

- there exists an index $\boldsymbol{v}^* > \bar{0}$ such that $\boldsymbol{v}_H = \mathfrak{I}(H) \leq \top - \boldsymbol{v}^*$ (*i.e.*, $\boldsymbol{v}_H \neq \top$);
- it is possible to compute $\mathfrak{L}(\Gamma_{\mathsf{aux}}(C))$, for any C and aux if τ is known;
- the DDH assumption holds in the group $\mathfrak{G}_{\boldsymbol{v}^*}$ of graded elements of index $\boldsymbol{v}^*$.

The two last conditions are only required for the smoothness to hold, and the last condition, the DDH assumption, can be relaxed to the κ-Lin assumption, as shown in Appendix F.3.

We remark these three conditions hold for quite a lot of SPHFs. They holds obviously for the KV-SPHF on Cramer-Shoup ciphertexts (example of Section 4.2). And they also hold, in particular, for our KV-SPHF for multi-exponentation equations cited in Section 6.2 and described in Appendix C.1, when all $A_{k,j}$ are in crs and can be generated in such a way we know their discrete logarithm (to be able to compute $\mathfrak{L}(\Gamma)$), and when the group $\mathbb{G}$ is replaced by a group $\mathbb{G}_1$ with $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ an asymmetric bilinear group (and DDH holds in $\mathbb{G}_2$). In both cases, $\mathfrak{G}_{\boldsymbol{v}_H} = \mathbb{G}_1$ and $\mathfrak{G}_{\boldsymbol{v}^*} = \mathbb{G}_2$.

Intuitively, our TSPHF construction is such that all the "SPHF" part of the TSPHF is in $\mathbb{G}_{\boldsymbol{v}_H}$, whereas the trapdoor part is in $\mathbb{G}_{\boldsymbol{v}^*}$. And the trapdoor part simply contains some representation of $\boldsymbol{\alpha}$, representation which cannot be used without knowing the trapdoor τ'.

The second CRS is a random graded ring element $\mathsf{crs}' = \zeta$ of index $\boldsymbol{v}^*$, and its trapdoor is $\tau' = \mathfrak{L}(\zeta)$. The hashing key $\mathsf{hk} = \boldsymbol{\alpha}$ is the same as before. The projection key is the ordered pair $\mathsf{hp} = (\boldsymbol{\gamma}, \boldsymbol{\chi})$, where $\boldsymbol{\gamma}$ is the same as before, and $\boldsymbol{\chi} = \zeta \cdot \boldsymbol{\alpha}$. The projection key is valid (*i.e.*, $\mathsf{VerHP}(\mathsf{hp}, \mathsf{full\text{-}aux}, C) = 1$) if and only if

$$\boldsymbol{\chi} \in \mathfrak{G}_{\boldsymbol{v}^*}^n \qquad \text{and} \qquad \zeta \cdot \boldsymbol{\gamma} = \Gamma \cdot \boldsymbol{\chi}, \tag{2}$$

Then, for any word $C \in \mathrm{L}_{\mathsf{full\text{-}aux}}$ with witness w corresponding to the vector $\boldsymbol{\lambda}$, the hash value is

$$H = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, C) \stackrel{\mathsf{def}}{=} \quad 1_{\boldsymbol{v}^*} \cdot \Theta(C) \cdot \boldsymbol{\alpha} = 1_{\boldsymbol{v}^*} \cdot \boldsymbol{\lambda} \cdot \boldsymbol{\gamma} \quad \stackrel{\mathsf{def}}{=} \mathsf{ProjHash}(\mathsf{hp}, \mathsf{full\text{-}aux}, C, w) = H', \tag{3}$$

where $1_{\boldsymbol{v}^*} = [\boldsymbol{v}^*, 1]$. Equation (2) means that $\boldsymbol{\chi}$ can be written $\boldsymbol{\chi} = \mathfrak{L}(\zeta) \cdot 1_{\boldsymbol{v}^*} \cdot \boldsymbol{\alpha'}$, with $\boldsymbol{\alpha'} \in \mathbb{Z}_p^n$ verifying $\boldsymbol{\gamma} = \Gamma \cdot \boldsymbol{\alpha'}$, *i.e.*, $\mathsf{hk}' = \boldsymbol{\alpha'}$ is a valid hashing key for $\boldsymbol{\gamma}$. We do not have necessarily $\boldsymbol{\alpha} = \boldsymbol{\alpha'}$, however, for any word $C \in \mathrm{L}_{\mathsf{full\text{-}aux}}$, we have and we set

$$H = 1_{\boldsymbol{v}^*} \cdot \Theta(C) \cdot \boldsymbol{\alpha} = 1_{\boldsymbol{v}^*} \cdot \boldsymbol{\lambda} \cdot \boldsymbol{\gamma} = 1_{\boldsymbol{v}^*} \cdot \Theta(C) \cdot \boldsymbol{\alpha'} = \mathfrak{L}(\zeta)^{-1} \cdot \Theta(C) \cdot \boldsymbol{\chi} \stackrel{\mathsf{def}}{=} \mathsf{THash}(\mathsf{hp}, \mathsf{full\text{-}aux}, C, \tau') = H''.$$

In Appendix E.3, we prove the resulting TSPHF is computationally smooth under the DDH assumption in $\mathfrak{G}_{\boldsymbol{v}^*}$, if $\mathfrak{L}(\Gamma_{\mathsf{aux}}(C))$ can be computed from τ. The correctness and the perfect soundness are easy to prove from the construction, and so the resulting TSPHF is correct, smooth and sound.

E.3 Smoothness of the Efficient TSPHF under DDH

In this section, we prove the computationally smoothness of the TSPHF described in Section E.2 (which is a formalization of the TSPHF described in Section 4.3), under the DDH assumption in $\mathfrak{G}_{\boldsymbol{v}^*}$, and assuming that τ is such that $\mathfrak{L}(\Gamma_{\mathsf{aux}}(C))$ can be computed efficiently. The computational smoothnees of our TSPHF can be reduced to the following computational assumption: it is hard to distinguish the two games (the one with $b = 0$ and the one with $b = 1$) depicted in Figure 10, where the procedure $\mathbf{O}$ can be called at most once. The games work as follows. The procedure **Initialize** generates and outputs the CRS crs. Then the adversary can make one query (aux, C) to the oracle $\mathbf{O}$ to get a tuple $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\gamma})$ such that:

- if $b = 0$, $\boldsymbol{\alpha} = \boldsymbol{\beta}$ is a random tuple of $\mathbb{Z}_p^n$, and $\boldsymbol{\gamma} = \Gamma \cdot \boldsymbol{\alpha}$, $\boldsymbol{A} = \boldsymbol{\alpha} \cdot \zeta$ and $\boldsymbol{B} = \boldsymbol{\beta} \cdot 1_{\boldsymbol{v}^*}$. In his case, we say that $(\zeta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\gamma})$ has been generated according to distribution 0;
- if $b = 1$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are random tuples of $\mathbb{Z}_p^n$ such that $\boldsymbol{\gamma} = \Gamma \cdot \boldsymbol{\alpha} = \Gamma \cdot \boldsymbol{\beta}$, $\boldsymbol{A} = \boldsymbol{\alpha} \cdot \zeta$ and $\boldsymbol{B} = \boldsymbol{\beta} \cdot 1_{\boldsymbol{v}^*}$. In this case, we say that $(\zeta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\gamma})$ has been generated according distribution 1.

Eventually, the adversary ends the game by querying the procedure **Finalize** with its guess b' for b.

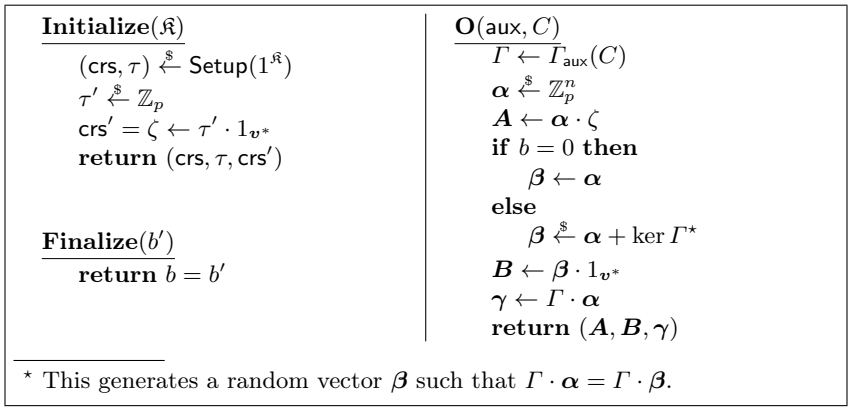

Fig. 10. Games used for the proof of smoothness under DDH ($b = 0$ or 1).

The reduction works as follows:

- on a $\mathbf{ProjKG}(\mathsf{hp}, C)$ query, we do as in the original smoothness game in Figure 1 except for the generation of hp, for which we call $\mathbf{O}$ to get a tuple $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\gamma})$ and set $\mathsf{hp} \leftarrow (\boldsymbol{\gamma}, \boldsymbol{A})$. Since we do not know $\boldsymbol{\alpha}$, we store $\boldsymbol{B}$ instead $\mathsf{hk} = \boldsymbol{\alpha}$.
- on a $\mathbf{Hash}(\mathsf{hp}, \mathsf{aux}, C)$, we do as in the original game except for the computation of H where we compute H as $H = \Theta(C) \cdot \boldsymbol{B}$. We remark that $H = 1_{\boldsymbol{v}^*} \cdot \Theta(C) \cdot \boldsymbol{\beta}$ and so, if $b = 0$, $H = 1_{\boldsymbol{v}^*} \cdot \Theta(C) \cdot \boldsymbol{\alpha}$ and H is computed as it would have been computed using hk. Otherwise, if $b = 1$:
  - if $C \notin \mathrm{L}_{\mathsf{full\text{-}aux}}$, $\mathfrak{L}(\Theta(C))$ is independent of the rows of $\mathfrak{L}(\Gamma)$ and $\mathfrak{L}(1_{\boldsymbol{v}^*} \cdot \Theta(C) \cdot \boldsymbol{\beta})$ and so $1_{\boldsymbol{v}^*} \cdot \Theta(C) \cdot \boldsymbol{\beta}$ is completely random given $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}$, and a fortiori given only hp;
  - otherwise, $\Theta(C) \cdot \boldsymbol{\alpha} = \Theta(C) \cdot \boldsymbol{\beta}$, because $\Gamma \cdot \boldsymbol{\alpha} = \Gamma \cdot \boldsymbol{\beta}$, and so H is computed as it would have been computed using hk.

Let us now show that the above computational assumption can be reduced to the DDH problem, which will prove the computational smoothness of the TSPHF.

We first remark that, $\boldsymbol{\gamma}$ is completely determined by $\mathfrak{L}(\boldsymbol{\gamma})$ and can be computed from $\mathfrak{L}(\boldsymbol{\gamma})$ by multiplying it by some vector $(1_{\boldsymbol{v}_1}, \ldots, 1_{\boldsymbol{v}_k})^\intercal \in \mathfrak{G}^k$ (depending on Γ). Let G be the set of all possible $\boldsymbol{\gamma}$: $G = \{\boldsymbol{\gamma'} \in \mathfrak{G}^k \mid \exists \boldsymbol{\alpha'} \in \mathfrak{G}^n, \boldsymbol{\gamma'} = \Gamma \cdot \boldsymbol{\alpha'}\}$ and $\mathfrak{L}(G) = \{\mathfrak{L}(\boldsymbol{\gamma'}) \in \mathbb{Z}_p^k \mid \boldsymbol{\gamma'} \in G\}$. It is clear that $\boldsymbol{\gamma}$ is uniformly distributed in

G. Furthermore, for all $\boldsymbol{\gamma} \in \mathbb{G}$, there exists a matrix $\Delta_{\boldsymbol{\gamma}} \in \mathbb{Z}_p^{n\times(m+1)}$, with $m = n - k$, such that the solutions of the equation $\mathfrak{L}(\boldsymbol{\gamma}) = \mathfrak{L}(\Gamma) \cdot \boldsymbol{\alpha}$ (where $\boldsymbol{\alpha} \in \mathbb{Z}_p^n$ is the unknown) are the vectors $\Delta \cdot \tilde{\boldsymbol{\delta}}$, for $\boldsymbol{\delta} \in \mathbb{Z}_p^m$, where $\tilde{\boldsymbol{x}} = \begin{pmatrix} \boldsymbol{x} \\ [\bar{0}, 1] \end{pmatrix} \in \mathfrak{G}^{m+1}$, for any colon vector $\boldsymbol{x} \in \mathfrak{G}^m$ ($[\bar{0}, 1]$ is the unitary element in $\mathbb{Z}_p$).

Let (ζ, d, e) be an instance of the DDH problem in $\mathfrak{G}_{\boldsymbol{v}^*}$. Let us write $d = \delta \cdot \zeta$ and $e = \eta \cdot 1_{\boldsymbol{v}^*}$. We just need to show how to generate one tuple $(\zeta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\gamma})$ with distribution 0 if (ζ, d, e) is a DDH tuple, and distribution 1 otherwise. The self-randomisability of the DDH problem enables us to generate $(\boldsymbol{d}, \boldsymbol{e}) \in \mathfrak{G}_{\boldsymbol{v}^*}^{2m}$ a completely random tuple if (ζ, d, e) is not a DDH tuple and a tuple such that $\boldsymbol{\delta} = \boldsymbol{\eta}$ otherwise, where $\boldsymbol{d} = \boldsymbol{\delta} \cdot \zeta$ and $\boldsymbol{e} = \boldsymbol{\eta} \cdot 1_{\boldsymbol{v}^*}$.

Then to generate $(\zeta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\gamma})$, we just need to choose $\mathfrak{L}(\boldsymbol{\gamma}) \in \mathbb{Z}_p^k$ uniformly at random in $\mathfrak{L}(G)$ and to set $\boldsymbol{A} = \Delta_{\boldsymbol{\gamma}} \cdot \tilde{\boldsymbol{d}}$, $\boldsymbol{B} = \Delta_{\boldsymbol{\gamma}} \cdot \tilde{\boldsymbol{e}}$.

F More on TSPHFs

In this appendix, we give details on particular technical points related to TSPHFs. We first show the construction of TSPHFs using NIZK in details. We then show that the TSPHFs constructed using our generic framework (Section 4.3) are always such that τ enables to decide efficiently if a word C is in the language $\text{LOFC}_{\mathsf{full\text{-}aux}}$ or not. Finally, we present an extension of the construction of TSPHFs based on our generic framework, for which the smoothness property can be proven under DLin or κ-Lin, instead of DDH.

F.1 Construction of TSPHFs using NIZK

In this section, we give details of the construction of TSPHFs using NIZK, sketched in Section 4.1.

ENIZK. Groth-Sahai NIZK [GS08] are actually ENIZK: when the CRS is in the perfectly sound mode, it is possible to extract witnesses which are group elements (but not witnesses y which are scalars in $\mathbb{Z}_p$, for which it is only possible to extract g_1^y or g_2^y, where g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$). Therefore, a Groth-Sahai ENIZK for the knowledge of some scalars requires to use one variable for each bit of the scalar, and makes the ENIZK very inefficient.

Generic Construction. To transform any SPHF into a TSPHF, it is sufficient to add to hp an ENIZK proof of the knowledge ($\aleph$) of an hashing key hk such that hp is the projection key of hk. Such an hp is valid, if and only if the ENIZK proof is valid. The CRS crs' of the TSPHF contains a CRS σ_{Π} for the ENIZK. The trapdoor τ' is the extraction trapdoor of the ENIZK. To compute the hash value of a valid word C, knowing only $\mathsf{hp} = (\mathsf{hp}', \pi)$, it is sufficient to extract a hashing key hk from π and to use it.

Let us prove that this construction yields a valid TSPHF. The hash correctness of the new TSPHF directly comes from the correctness of the SPHF. The trapdoor correctness comes from the completeness and the extractability of the ENIZK. The soundness comes from the extractability of the ENIZK. Finally, the computational smoothness of the SPHF can be reduced to the smoothness of the TSPHF. The reduction consists in generating a simulated CRS for the ENIZK, and on a $\mathbf{ProjKG}(\mathsf{aux}, C)$ query, to get a projection key hp' for the original SPHF by calling the $\mathbf{ProjKG}$ oracle of the SPHF smoothness game and to return $\mathsf{hp} = (\mathsf{hp}', \pi)$, with π a simulated ENIZK that hp is valid.

Improved Constructions for our Generic SPHF Framework. In all our practical constructions, we remark that we can use Groth-Sahai methodology for ENIZK proofs. But the hashing key hk is a tuple of scalars. Thus the ENIZK has to be bit-by-bit and this makes it not very efficient.

In [JR12], the authors show that it is possible to avoid doing a bit-by-bit ENIZK, for the SPHF they use, if the hash value is slightly changed. This trick can be further extended to any SPHF based on our generic framework, as soon as H is in some group $\mathbb{G}_1$, where $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ is a bilinear group. The idea just consists in replacing the hash value H by $e(H, g_2)$ and to remark that, knowing $\boldsymbol{\alpha'} = g_2 \odot \boldsymbol{\alpha} = (g_2^{\alpha_1}, \ldots, g_2^{\alpha_n}) \in \mathbb{G}_2^n$ is sufficient to compute the hash value:

$$H = e(\Theta(\boldsymbol{u}) \odot \boldsymbol{\alpha}, g_2) = \Theta(\boldsymbol{u}) \odot \boldsymbol{\alpha} \odot g_2 = \Theta(\boldsymbol{u}) \odot \boldsymbol{\alpha'}.$$

Then, to transform this new SPHF into a TSPHF, it is sufficient to add, to $\mathsf{hp} = \boldsymbol{\gamma}$, an ENIZK proof that one knows $\boldsymbol{\alpha'}$ such that $\Gamma \odot \boldsymbol{\alpha'} = \boldsymbol{\gamma} \odot g_2$, *i.e.*, the logarithm of $\boldsymbol{\alpha'}$ is some $\boldsymbol{\alpha}$ such that $\mathsf{hk} = \boldsymbol{\alpha}$ is a valid hashing key. The Groth-Sahai methodology can be used to construct the ENIZK. Since $\boldsymbol{\alpha'}$ is a vector of group elements, the construction is very efficient.

Actually, the idea of "storing" the hashing key in $\mathbb{G}_2$ is one of the basic idea of our efficient TSPHF construction of Section 4.3. But the latter construction is a lot more efficient than the above construction because it only requires to add one group element to hp, for each α_i, instead of a Groth-Sahai ENIZK proof, which requires at least a commitment of one group element for each α_i, where each commitment requires two group elements.

F.2 Remark on τ

Our TSPHF construction requires that τ enables to compute efficiently the "discrete logarithm of $\Gamma(C)$" ($\mathfrak{L}(\Gamma(C))$). Let us show that this implies it is possible to know whether a word $C \in \mathcal{S}et$ is in $\mathrm{L}_{\mathsf{full\text{-}aux}}$ or not, using τ, with overwhelming probability.

Since $\mathfrak{L}(\Gamma)$ is known, it is possible to choose two random hashing keys $\mathsf{hk} = \boldsymbol{\alpha}$ and $\mathsf{hk}' = \boldsymbol{\alpha'}$ such that $\mathfrak{L}(\Gamma) \cdot (\boldsymbol{\alpha} - \boldsymbol{\alpha'}) = 0$, so that hk and hk' correspond to the same projection key. For any ciphertext $C \in \mathrm{L}_{\mathsf{aux}}$, the hash value of C under hk is the same as the hash value of C under hk'. And for any ciphertext $C \in \mathcal{S}et \setminus \mathrm{L}_{\mathsf{aux}}$, the hash values of C under hk and under hk' are two independent uniform random variables, and are equal with probability at most $1/p$.

This implies that the condition, in Section 7.1, that for any w and x, knowing τ provides a way to test whether x is valid and w is a partial witness of x, with overwhelming probability, is actually always verified for TSPHF generated using the method of Section 4.3.

F.3 Extension of the TSPHF Construction to κ-Lin

We present an extension of the construction of TSPHFs based on our generic framework, for which the smoothness property can be proven under DLin or κ-Lin, instead of DDH. We use the notations of the full generic framework in Appendix E.

DLin and κ-Lin Assumptions. Before showing this extension of TSPHF, let us first recall the DLin and the κ-Lin assumptions.

Definition 3 (Decisional Linear Problem (DLin)). *The Decisional Linear Problem [BBS04] says that, in a group $(p, \mathbb{G}, g)$, when we are given $(g^x, g^y, g^{xa}, g^{yb}, g^c)$ for unknown random $x, y, a, b \xleftarrow{\$} \mathbb{Z}_p$, it is hard to decide whether $c = a + b \bmod p$ (a linear tuple) or $c \xleftarrow{\$} \mathbb{Z}_p$ (a random tuple). We define by $\mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t)$ the best advantage an adversary can have in distinguishing a linear tuple from a random tuple within time t.*

The latter problem has been generalized into the κ-Lin [HK07, Sha07]:

Definition 4 (Decisional κ-Linear Problem (κ-Lin)). *The Decisional κ-Linear Problem says that, in a group $(p, \mathbb{G}, g)$, when we are given $(g^{x_1}, \ldots, g^{x_\kappa}, g^{x_1 y_1}, \ldots, g^{x_\kappa y_\kappa}, g^z)$ for unknown random $x_i, y_i \xleftarrow{\$} \mathbb{Z}_p$, $i = 1, \ldots, \kappa$, it is hard to decide whether $z = \sum y_i \bmod p$ (a κ-linear tuple) or $z \xleftarrow{\$} \mathbb{Z}_p$ (a random tuple). We define by $\mathsf{Adv}^{\kappa\text{-lin}}_{p,\mathbb{G},g}(t)$ the best advantage an adversary can have in distinguishing a linear tuple from a random tuple within time t.*

One can note that 1-Lin = DDH and 2-Lin =DLin, and also, the larger is κ, the weaker is the κ-Lin assumption.

Construction. The CRS crs' contains κ random elements $\zeta_1, \ldots, \zeta_\kappa$ of index $\boldsymbol{v}^*$. The trapdoor τ' is the tuple of the discrete logarithms of these elements: $\tau' = (\mathfrak{L}(\zeta_1), \ldots, \mathfrak{L}(\zeta_\kappa))$. The idea is to duplicate hk and hp κ-time and to use the product of the hash values for all these keys as hash value.

More precisely, the hashing key hk is a tuple of κ hashing key of the original scheme $\boldsymbol{\alpha}_l$ (for $l = 1, \ldots, \kappa$) *i.e.*, $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_\kappa$ are independant random vectors of $\mathbb{Z}_p^n$ (we write $\alpha_{l,i}$ for the ith coefficient of $\boldsymbol{\alpha}_l$). The projection key hp is the κ-tuple of the corresponding projection keys $\mathsf{hp}_l = (\boldsymbol{\gamma}_l, \boldsymbol{\chi}_l)$ (for $l \in \{1, \ldots, \kappa\}$). It can be verified by verifying the κ projections keys $\mathsf{hp}_1, \ldots, \mathsf{hp}_\kappa$.

Then the hash value is:

$$H = \mathsf{Hash}(\mathsf{hk}, \mathsf{full\text{-}aux}, \boldsymbol{u}) \stackrel{\mathsf{def}}{=} \sum_{l=1}^{\kappa} 1_{\boldsymbol{v}^*} \cdot \Theta(\boldsymbol{u}) \cdot \boldsymbol{\alpha}_l = \sum_{l=1}^{\kappa} 1_{\boldsymbol{v}^*} \cdot \boldsymbol{\lambda}_l^{\mathsf{T}} \cdot \boldsymbol{\gamma}_l \stackrel{\mathsf{def}}{=} \mathsf{ProjHash}(\mathsf{hp}, \mathsf{full\text{-}aux}, w) = H'.$$

This value can also be computed using the trapdoor τ' and $\boldsymbol{\chi}$:

$$H' = \sum_{l=1}^{\kappa} \mathfrak{L}(\zeta_l)^{-1} \cdot \Theta(\boldsymbol{u}) \cdot \boldsymbol{\chi}_l \stackrel{\mathsf{def}}{=} \mathsf{THash}(\tau', \mathsf{hp}, \mathsf{full\text{-}aux}, \bot) = H''.$$

Proof of Security. The proof is very similar to the DDH case. Figure 11 shows the main games in the proof.

Initialize$(\Re)$	**O**(aux, C)
$(\mathsf{crs}, \tau) \xleftarrow{\$} \mathsf{Setup}(1^\Re)$	$\mathsf{full\text{-}aux} \leftarrow (\mathsf{crs}, \mathsf{aux})$
$\tau' = (t_1, \ldots, t_\kappa) \xleftarrow{\$} \mathbb{Z}_p^\kappa$	$\Gamma \leftarrow \Gamma_{\mathsf{aux}}(C)$
$\mathsf{crs}' = (\zeta_1, \ldots, \zeta_\kappa) \leftarrow (t_1 \cdot 1_{\boldsymbol{v}^*}, \ldots, t_\kappa \cdot 1_{\boldsymbol{v}^*})$	**for** $l = 1, \ldots, \kappa$ **do**
$b \xleftarrow{\$} \{0, 1\}$	$\boldsymbol{\alpha}_l \xleftarrow{\$} \mathbb{Z}_p^n$
return $(\mathsf{crs}, \tau, \mathsf{crs}')$	$\boldsymbol{A}_l \leftarrow \boldsymbol{\alpha}_l \cdot \zeta_l$
Finalize(b')	**if** $b = 0$ **then**
return $b = b'$	$\boldsymbol{\beta} \leftarrow \boldsymbol{\alpha}_1 + \cdots + \boldsymbol{\alpha}_\kappa$
	else
	$\boldsymbol{\beta} \xleftarrow{\$} \boldsymbol{\alpha}_1 + \cdots + \boldsymbol{\alpha}_\kappa + \ker \Gamma$
	$\boldsymbol{B} \leftarrow \boldsymbol{\beta} \cdot 1_{\boldsymbol{v}^*}$
	$\boldsymbol{\gamma} \leftarrow \Gamma \cdot \boldsymbol{\alpha}$
	return $(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\kappa, \boldsymbol{B}, \boldsymbol{\gamma})$

Fig. 11. Games used for the proof of smoothness under κ-Lin ($b = 0$ or 1).