Updatable Trapdoor SPHFs: Modular Construction of Updatable Zero-Knowledge Arguments and More[⋆]

Behzad Abdolmaleki[1] and Daniel Slamanig[2]

[1] Max Planck Institute for Security and Privacy, Bochum, Germany
behzad.abdolmaleki@csp.mpg.de
[2] AIT Austrian Institute of Technology, Vienna, Austria
daniel.slamanig@ait.ac.at

Abstract. Recently, motivated by its increased use in real-world applications, there has been a growing interest on the reduction of trust in the generation of the common reference string (CRS) for zero-knowledge (ZK) proofs. This line of research was initiated by the introduction of subversion non-interactive ZK (NIZK) proofs by Bellare *et al.* (ASIACRYPT'16). Here, the zero-knowledge property needs to hold even in case of a malicious generation of the CRS. Groth *et al.* (CRYPTO'18) then introduced the notion of updatable zk-SNARKS, later adopted by Lipmaa (SCN'20) to updatable quasi-adaptive NIZK (QA-NIZK) proofs. In contrast to the subversion setting, in the updatable setting one can achieve stronger soundness guarantees at the cost of reintroducing some trust, resulting in a model in between the fully trusted CRS generation and the subversion setting. It is a promising concept, but all previous updatable constructions are ad-hoc and tailored to particular instances of proof systems. Consequently, it is an interesting question whether it is possible to construct updatable ZK primitives in a more modular way from simpler building blocks.

In this work we revisit the notion of trapdoor smooth projective hash functions (TSPHFs) in the light of an updatable CRS. TSPHFs have been introduced by Benhamouda *et al.* (CRYPTO'13) and can be seen as a special type of a 2-round ZK proof system. In doing so, we first present a framework called *lighter* TSPHFs (*L-TSPHFs*). Building upon it, we introduce updatable *L-TSPHF*s as well as instantiations in bilinear groups. We then show how one can generically construct updatable quasi-adaptive zero-knowledge arguments from updatable *L-TSPHFs*. Our instantiations are generic and more efficient than existing ones. Finally, we discuss applications of (updatable) *L-TSPHFs* to efficient (updatable) 2-round ZK arguments as well as updatable password-authenticated key-exchange (uPAKE).

1 Introduction

Zero-knowledge (ZK) proofs were introduced by Goldwasser, Micali and Rackoff [GMR89] and play a central role in both the theory and practice of cryptography. A long line of research [Kil92, GOS06, GS08, Gro10, Lip12, GGPR13, JR13,

[⋆] This is the full version of a paper which appears in the proceedings of the 26th Australasian Conference on Information Security and Privacy - ACISP 2021, LNCS, Springer.

JR14, KW15, Gro16] has led to efficient pairing-based zero-knowledge Succint Non-interactive ARguments of Knowledge (zk-SNARKs) and succinct Quasi-Adaptive Non-Interactive Zero-Knowledge arguments (QA-NIZKs) in the common reference string (CRS) model. QA-NIZKs are a relaxation of NIZK proofs where the CRS is allowed to depend on the specific language for which proofs have to be generated [JR13, LPJY14, JR14, KW15, LPJY15, AJOR18]. In general, SNARKs (QA-NIZKs) are succinct, in fact, they allow to prove that circuits of arbitrary size (for linear languages) are satisfied with a constant-size proof. They are also concretely very efficient, 3 group elements is the best SNARK construction for arithmetic circuits [Gro16] and 1 group element is the best QA-NIZK construction for linear languages [KW15]. Recently, Campanelli *et al.* [CFQ19] proposed LegoSNARK, a toolbox for commit-and-prove zk-SNARKs (CP-SNARKs), where they use succinct QA-NIZKs as efficient zk-SNARKs for linear subspace languages.

Trust in the CRS. For the practical application of zero-knowledge primitives, an important question is the generation of the CRS. While in theory it is simply assumed that some trusted party will perform the CRS generation, such a party is hard to find in the real-world. Recently, there has been an increasing interest to reduce the trust in the generator of the CRS. Existing approaches are (1) the use of multi-party computation to generate the CRS in a distributed way [BSCG$^+$15, BGG17, ABL$^+$19] or (2) the use of CRS checking algorithms in subversion NIZKs [BFS16], zk-SNARKS [ABLZ17, Fuc18] and QA-NIZKs [ALSZ20]. Here, although the prover does not need to trust the CRS, the zero-knowledge property (so called subversion ZK) is still preserved. However, the verifier still needs to trust the CRS generator. Abdolmaleki *et al.* [ALSZ20] later studied the Kiltz-Wee QA-NIZKs [KW15] in a variant of the bare public-key (BPK) model, where some part of the CRS (the language parameter) is generated by a trusted party, but the rest of the CRS can be generated maliciously by some untrusted party (from the prover's perspective). Finally, (3) there is the recent approach of a so called *updatable CRS* [GKM$^+$18, MBKM19, DRZ20, CHM$^+$20, Lip20, ARS20]. Here, everyone can update a CRS such that the updates can be verified and ZK holds in front of a malicious CRS generator and the verifier can trust the CRS (soundness holds) as long as one operation, either the CRS creation or one of its updates, have been performed honestly. So a verifier can do a CRS update on its own and then send the updated CRS to the prover. Note that this updating inherently requires communication of the prover and the verifier in such an updatable SNARK/QA-NIZK setting, a fact that will be useful for us.

QA-ZK. In the following, we focus on ZK in the quasi-adaptive setting. QA-NIZK were introduced by Jutla and Roy in [JR13] and further improved in e.g., [JR14, KW15]. Jutla and Roy have shown that for linear languages (linear subspaces of vector spaces over bilinear groups), one can obtain more efficient computationally-sound NIZK proofs (so called arguments) when compared to Groth-Sahai proofs [GS08]. They are in a slightly different quasi-adaptive setting, which however suffices for many cryptographic applications and can be particularly useful in generic toolboxes such as LegoSNARK [CFQ19]. In the quasi-adaptive setting, a class of parametrized languages $\{\mathtt{lpar}\}$ is considered and the CRS generator is allowed to generate the CRS based on the language parameter $\mathtt{lpar}$. As already mentioned, recently, Abdolmaleki *et al.* [ALSZ20] made the Kiltz-Wee QA-NIZK [KW15] subversion resistant and also showed that this construction is equivalent to 2-round ZK arguments or more general it is a QA-NIZK in a variant of the bare public-key (BPK) model. Then Lipmaa [Lip20] proposed an updatable version of the QA-NIZK construction of [ALSZ20]. However, all these constructions are ad-hoc based on and for specific proof systems and not generic.

(QA)-ZK and SPHFs. Smooth Projective Hash Functions (SPHFs) [CS02] (cf. Section 2) can be viewed as honest-verifier zero-knowledge (HVZK) arguments for the membership in specific languages [ACP09]. HVZK is a weakened variant of ZK, which only needs to hold for honest verifiers. Roughly speaking, to prove membership of $\mathtt{x} \in \mathcal{L}$, the verifier generates the secret hashing key hk, and for any word $\mathtt{x}$ she can compute the hash value H without knowledge of the witness $\mathtt{w}$ by using the hashing key hk. In addition, the verifier can derive a projection key hp from the hashing key hk and send it to the prover. By knowing a witness $\mathtt{w}$ for membership of $\mathtt{x} \in \mathcal{L}$ and having the projection key hp, the prover is able to efficiently compute the projective hash pH for the word $\mathtt{x}$ such that it equals the hash H computed by the verifier. The smoothness property implies that if $\mathtt{x} \notin \mathcal{L}$, one cannot guess the hash value H by knowing hp, or in other words, the hash value H looks completely random. Benhamouda *et al.* [BBC+13, BP13] showed how one can construct ZK instead of HVZK arguments in the CRS model by introducing so called Trapdoor SPHFs (TSPHFs). Recently, Abdolmaleki *et al.* [AKL21] defined a variant of TSPHF with an untrusted setup, so called smooth zero-knowledge hash functions, and show how to construct 2-round ZK arguments in the plain (or subversion ZK arguments in the CRS) model under a non-falsifiable assumption. Also Abdalla *et al.* in [ABP15] proposed a framework for QA-(NI)ZK based on the disjunction of two languages and SPHFs and provided an alternative view of the Kiltz-Wee QA-NIZK construction. Compared with the ZK arguments (or QA-NIZK in the BPK model) in [ALSZ20], the QA-ZK arguments based on TSPHFs in [BBC+13, BP13] are less efficient regarding proof size, computation and communication complexity. Moreover, it does not yield a modular construction for updatable QA-ZK, a gap that we close.

Our Contribution and Technical Overview. This work is motivated by the lack of modular and simple building blocks to construct updatable ZK primitives. We address this in this works as follows:

Lighter TSPHFs. We first revisit the notion of TSPHFs proposed by Benhamouda *et al.* [BBC+13, BP13], which represents an extension of a classical SPHF, and requires that the setup algorithm $\mathsf{Pgen}(1^\lambda)$ outputs an additional CRS $\mathtt{crs}'$ and a trapdoor τ' specific to $\mathtt{crs}'$. This trapdoor can be used by thash (trapdoor hashing) to compute the hash value of words $\mathtt{x}$ knowing only hp. This is useful to allow simulation in the construction of ZK protocols. We present a new approach which we call lighter TSPHFs (L-TSPHFs), allowing instantiations in bilinear groups that are more efficient than known TSPHFs, as all three hashing algorithms hash, $\mathsf{projhash}$, and thash yield hash values in $\mathbb{G}_1$ instead of $\mathbb{G}_T$. Our L-TSPHF framework that forms the basis for our TSPHF framework with an updatable CRS (denoted *uL-TSPHF*), is parametrized by a SPHF Σ and additionally relies on a new knowledge assumption $\mathsf{LTSPHF\text{-}KE}$ (which we will prove based on Hash-Algebraic Knowledge (HAK) assumptions [Lip19])[3]. We stress that our main motivation for *L-TSPHFs* is the construction of *uL-TSPHF* and updatable QA-ZK proofs. As from [GKM+18, Lip20] it is known that in order to have updatability, knowledge assumptions are crucial for extracting the new trapdoor from an updated CRS, as it might have been updated by a dishonest party, this does not represent a limitation.

Updatable Lighter TSPHFs and QA-ZK. We present a framework for updatable *L-TSPHF*s (*uL-TSPHF*s) which is inspired by updatable SNARKs [GKM+18, MBKM19, CHM+20, ARS20] and updatable QA-NIZKs [Lip20]. In short, we add algorithms crsVer for checking the well-formedness of the CRS, Upd for performing CRS updates (outputting a proof of correct update) and UpdVer for checking the correctness of an update by means of update proofs and define the security requirements for *uL-TSPHF*s. In contrast to the ad-hoc constructions of updatable SNARKs [GKM+18, MBKM19, CHM+20] and updatable QA-NIZKs [Lip20], our updatable *L-TSPHF* framework is a generic building block which can be used to modularly design updatable primitives. Our instantiation of an *uL-TSPHF* is directly based on an *L-TSPHF* together with a suitable additive updating procedure of the trapdoor in the CRS and extraction based on the BDH-knowledge assumption [ABLZ17], representing a simple variant of the PKE assumption [DFGK14].We then show as the main application of *uL-TSPHF*s the construction of updatable QA-ZK arguments. When compared with the only existing construction of updatable QA-ZK proofs in [Lip20] (which is ad-hoc), we can significantly reduce the proof as well as the communication size and in particular obtain succinct proofs.

Applications. Besides updatable QA-ZK, we provide an application of the *L-TSPHF* framework for constructing QA-ZK arguments[4] in a modular way. Using our instantiations under the LTSPHF-KE assumption in bilinear groups, we show that *L-TSPHFs* yield a framework for constructing efficient 2-round ZK arguments with a pairing-free verifier. The resulting ZK arguments are more efficient than previous QA-ZK constructions in [BP13, ALSZ20]. We also present a concrete instance for proving the correct encryption of a valid Waters signature. Finally, as another interesting application, we show how to construct an updatable two-round Password-Authenticated Key-Exchange (uPAKE) protocol from *uL-TSPHF*s, which allows to reduce trust in the setup of the PAKE.

2 Preliminaries

Let PPT denote probabilistic polynomial-time. Let $\lambda \in \mathbb{N}$ be the security parameter. All adversaries will be stateful. For an algorithm A, $\mathsf{RND}(\mathsf{A})$ is the random tape of A (for a fixed choice of λ), and $\omega \leftarrow_{\$} \mathsf{RND}(\mathsf{A})$ denotes the random choice of ω

[3] HAK is essentially a concrete Algebraic Group Model (AGM) [FKL18] version of the generic group model with hashing that models the ability of an adversary to create elliptic-curve group elements by using elliptic-curve hashing without knowing their discrete logarithm.

[4] We note that all ZK arguments we consider in this paper are in the quasi-adaptive setting and we might sometimes omit to make this explicit.

from $\mathsf{RND}(\mathsf{A})$. By $x \leftarrow_{\$} \mathcal{D}$ we denote that x is sampled according to distribution $\mathcal{D}$ or uniformly randomly if $\mathcal{D}$ is a set. A bilinear group generator $\mathsf{BG.Pgen}(1^\lambda)$ returns $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \bar{e})$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive cyclic groups of prime order p, and $\bar{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate efficiently computable bilinear pairing. We use the implicit bracket notation of [EHK+13], that is, we write $[a]_\iota$ to denote ag_ι where g_ι is a fixed generator of $\mathbb{G}_\iota$. We denote $\bar{e}([a]_1, [b]_2)$ as $[a]_1 \cdot [b]_2$. Thus, $[a]_1 \cdot [b]_2 = [ab]_T$ (also $[a]_2 \cdot [b]_1 = [ab]_T$). We denote $s[a]_\iota = [sa]_\iota$ for $s \in \mathbb{Z}_p$ and $S \cdot [a]_\iota = [Sa]_T$ for $S \in \mathbb{G}_\iota$ and $\iota \in \{1, 2, T\}$. We freely use the bracket notation together with matrix notation, for example, if $\mathbf{AB} = \mathbf{C}$ then $[\mathbf{A}]_1 \cdot [\mathbf{B}]_2 = [\mathbf{C}]_T$.

Smooth Projective Hash Functions. Smooth projective hash functions (SPHF) [CS02] are families of pairs of functions $(\mathsf{hash}, \mathsf{projhash})$ defined on a language $\mathcal{L}$. They are indexed by a pair of associated keys $(\mathsf{hk}, \mathsf{hp})$, where the hashing key hk may be viewed as the private key and the projection key hp as the public key. On a word $\mathtt{x} \in \mathcal{L}$, both functions need to yield the same result, that is, $\mathsf{hash}(\mathsf{hk}, \mathcal{L}, \mathtt{x}) = \mathsf{projhash}(\mathsf{hp}, \mathcal{L}, \mathtt{x}, \mathtt{w})$, where the latter evaluation additionally requires a witness $\mathtt{w}$ that $\mathtt{x} \in \mathcal{L}$. Thus, they can be seen as a tool for implicit designated-verifier proofs of membership [ACP09]. Formally SPHFs are defined as follows (cf. [BBC+13]).

Definition 1 (SPHF). *A SPHF for a language $\mathcal{L}$ is a tuple of PPT algorithms* $(\mathsf{Pgen}, \mathsf{hashkg}, \mathsf{projkg}, \mathsf{hash}, \mathsf{projhash})$, *which are defined as follows:*

$\mathsf{Pgen}(1^\lambda, \mathcal{L})$**.** *Takes the security parameter λ and the language $\mathcal{L}$, and generates the language parameters* $\mathtt{lpar}$.

$\mathsf{hashkg}(\mathcal{L})$**:** *Takes a language $\mathcal{L}$ and outputs a hashing key* hk *for $\mathcal{L}$.*

$\mathsf{projkg}(\mathsf{hk}, \mathtt{lpar}, \mathtt{x})$**:** *Takes a hashing key* hk*, a language parameter* $\mathtt{lpar}$*, and a word* $\mathtt{x}$ *and outputs a projection key* hp*, possibly depending on* $\mathtt{x}$.

$\mathsf{hash}(\mathsf{hk}, \mathtt{lpar}, \mathtt{x})$**:** *Takes a hashing key* hk*, a language parameter* $\mathtt{lpar}$*, and a word* $\mathtt{x}$ *and outputs a hash* H.

$\mathsf{projhash}(\mathsf{hp}, \mathtt{lpar}, \mathtt{x}, \mathtt{w})$**:** *Takes a projection key* hp*, a language parameter* $\mathtt{lpar}$*, a word* $\mathtt{x}$*, and a witness* $\mathtt{w}$ *for* $\mathtt{x} \in \mathcal{L}$ *and outputs a hash* pH.

A SPFH needs to satisfy the following properties:

Correctness. It is required that $\mathsf{hash}(\mathsf{hk}, \mathtt{lpar}, \mathtt{x}) = \mathsf{projhash}(\mathsf{hp}, \mathtt{lpar}, \mathtt{x}, \mathtt{w})$ for all $\mathtt{x} \in \mathcal{L}$ and their corresponding witnesses $\mathtt{w}$.

Smoothness. It is required that if $\mathtt{x} \notin \mathcal{L}$, the following distributions are statistically indistinguishable:

$$\left\{ (\mathcal{L}, \mathtt{x}, \mathsf{hp}, \mathsf{H}) : \begin{array}{l} \mathtt{lpar} \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L}), \mathsf{hk} \leftarrow \mathsf{hashkg}(\mathcal{L}), \\ \mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk}, \mathtt{lpar}, \mathtt{x}), \mathsf{H} \leftarrow \mathsf{hash}(\mathsf{hk}, \mathtt{lpar}, \mathtt{x}) \end{array} \right\},$$

$$\left\{ (\mathcal{L}, \mathtt{x}, \mathsf{hp}, \mathsf{H}) : \begin{array}{l} \mathsf{pars} \leftarrow \mathsf{Pgen}(1^\lambda), \mathsf{hk} \leftarrow \mathsf{hashkg}(\mathcal{L}), \\ \mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk}, \mathtt{lpar}, \mathtt{x}), \mathsf{H} \leftarrow_{\$} \Pi \end{array} \right\},$$

where the range Π is the set of hash values.

Depending on the definition of smoothness, there are three types of SPHFs (cf. [BBC$^+$13]):

GL-SPHF. The projection key hp can depend on word $\mathtt{x}$ and so the smoothness is correctly defined only if $\mathtt{x}$ is chosen before having seen hp.
KV-SPHF. hp does not depend on word $\mathtt{x}$ and the smoothness holds even if $\mathtt{x}$ is chosen after having seen hp.
CS-SPHF. hp does not depend on word $\mathtt{x}$ but the smoothness holds only if $\mathtt{x}$ is chosen before having seen hp.

Language Representation. Similar to [BBC$^+$13], for a language $\mathcal{L}$, we assume there exist two positive integers k and n, a function $\mathbf{\Gamma} : S \to \mathbb{G}^{k \times n}$, and a family of functions $\mathbf{\Theta} : S \to \mathbb{G}^{1 \times n}$, such that for any word $\mathtt{x} \in S$, $(\mathtt{x} \in \mathcal{L})$ iff $\exists \lambda \in \mathbb{Z}_p^{1 \times k}$ such that $\mathbf{\Theta}(\mathtt{x}) = \lambda \mathbf{\Gamma}(\mathtt{x})$. In other words, we assume that $\mathtt{x} \in \mathcal{L}$, if and only if, $\mathbf{\Theta}(\mathtt{x})$ is a linear combination of (the exponents in) the rows of some matrix $\mathbf{\Gamma}(\mathtt{x})$. For a KV-SPHF, $\mathbf{\Gamma}$ is supposed to be a constant function (independent of the word $\mathtt{x}$), otherwise one obtains a GL-SPHF. We furthermore require that, when knowing a witness $\mathtt{w}$ of the membership $\mathtt{x} \in \mathcal{L}$, one can efficiently compute the above linear combination λ. This may seem a quite strong requirement, but this is satisfied by very expressive languages over ciphertexts such as ElGamal, Cramer-Shoup (CS) and variants.

Trapdoor Smooth Projective Hash Functions. Benhamouda *et al.* proposed an extension of a classical SPHF, called TSPHF [BBC$^+$13]. Their framework has an additional algorithm $\mathsf{Pgen}(1^\lambda)$ outputs an additional CRS $\mathtt{crs}'$ and a trapdoor τ' specific to $\mathtt{crs}'$, which can be used to compute the hash value of words $\mathtt{x}$ knowing only hp.

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

Definition 2 (TSPHF [BBC$^+$13]). *A TSPHF for a language $\mathcal{L}$ is defined by seven algorithms:*

- $\mathsf{Pgen}(1^\lambda, \mathcal{L})$**.** *Takes as input the security parameter λ and the language $\mathcal{L}$ and generates the language parameter* $\mathtt{lpar}$*, the CRS* $\mathtt{crs}'$*, together with a trapdoor τ'.*
- hashkg, projkg, hash, *and* $\mathsf{projhash}$, *are the same as for a classical SPHF.*
- $\mathsf{verhp}(\mathsf{hp}, \mathtt{lpar}, \mathtt{crs}', \mathtt{x})$**.** *Takes a language* hp, $\mathtt{lpar}$, $\mathtt{crs}'$, *and the word* $\mathtt{x}$ *and outputs 1 if* hp *is a valid projection key, and 0 otherwise.*
- $\mathsf{thash}(\mathsf{hp}, \mathtt{lpar}, \mathtt{crs}', \mathtt{x}, \tau')$**:** *Takes a hashing key* $\mathtt{lpar}$, $\mathtt{crs}'$, *the word* $\mathtt{x}$, *and the trapdoor τ', and outputs the hash value of* $\mathtt{x}$ *from the projection key* hp *and the trapdoor τ'.*

There is an additional property on the language $\mathcal{L}$ that it has to be witness sampleable. By witness sampleable, we mean that there exists a trapdoor $\mathtt{tc}_{\mathtt{lpar}}$ for the language parameters $\mathtt{lpar}$, such that $\mathtt{tc}_{\mathtt{lpar}}$ enables to efficiently compute the discrete logarithms of the entries of $\mathtt{lpar}$. A TSPHF must satisfy the following properties:

Correctness. For any word $\mathtt{x} \in \mathcal{L}$ with witness $\mathtt{w}$, for any $\mathsf{hk} \leftarrow \mathsf{hashkg}(\mathcal{L})$ and for

```
$\mathsf{Exp}^{\mathsf{smooth}-\mathsf{b}}(\mathcal{A}, \lambda)$

– $(\mathtt{lpar}, \mathtt{crs}', \tau') \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L}), \mathsf{hk} \leftarrow \mathsf{hashkg}(\mathcal{L}, \mathtt{lpar}), \mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk}, \mathcal{L}, \mathtt{lpar})$
– $\mathtt{x} \leftarrow \mathcal{A}(\mathtt{lpar})$
– If $b = 0$ or $\mathtt{x} \in \mathcal{L}$, then $\mathsf{H} \leftarrow \mathsf{hash}(\mathsf{hk}, \mathtt{lpar}, \mathcal{L}, \mathtt{x})$, else $\mathsf{H} \leftarrow_\$ \Pi$.
– **return** $\mathcal{A}(\mathtt{lparx}, \mathsf{hp}, \mathsf{H})$.
```

Fig. 1. Experiments $\mathsf{Exp}^{\mathsf{smooth}-\mathsf{b}}$ for computational smoothness

$\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk}, \mathcal{L}, \mathtt{x})$ it should satisfy the two properties: *hash correctness*, and *trapdoor correctness*. The fist property corresponds to correctness for classical SPHFs, and the second one states that $\mathsf{verhp}(\mathsf{hp}, \mathtt{lpar}, \mathtt{crs}', \mathtt{x}) = 1$ and $\mathsf{hash}(\mathsf{hk}, \mathtt{lpar}, \mathtt{crs}', \mathtt{x}) = \mathsf{thash}(\mathsf{hp}, \mathtt{lpar}, \mathtt{crs}', \mathtt{x})$, with overwhelming probability.

(t, ϵ)-soundness property. Given $\mathtt{lpar}$, its trapdoor, and $\mathcal{L}$ and $\mathtt{crs}'$, no adversary running in time at most t can produce a projection key hp, a value aux, a word $\mathtt{x}$ and valid witness $\mathtt{w}$ such that $\mathsf{verhp}(\mathsf{hp}, \mathtt{lpar}, \mathtt{crs}', \mathtt{x}) = 1$ but $\mathsf{projhash}(\mathtt{w}, \mathtt{lpar}, \mathtt{crs}', \mathtt{x}) \neq \mathsf{thash}(\mathsf{hp}, \mathtt{lpar}, \mathtt{crs}', \mathtt{x})$ with probability at least ϵ.

Smoothness. Is the same as for SPHFs, except that, Pgen outputs extra elements τ' and $\mathtt{crs}'$, but while the trapdoor τ' of the $\mathtt{crs}'$ is dropped, $\mathtt{crs}'$ is forwarded to the adversary (together with the language parameter $\mathtt{lpar}$).

Notice that since hp now needs to contain enough information to compute the hash value of any word $\mathtt{x}$, the smoothness property of TSPHFs is no longer statistical but computational. The computational smoothness is defined by the experiments $\mathsf{Exp}^{smooth-b}$ and depicted in Fig. 1.

Quasi-Adaptive Zero-Knowledge Arguments. A tuple of PPT algorithms $\Pi = (\mathsf{Pgen}, \mathsf{K}_{\mathsf{crs}}, \mathsf{P}, \mathsf{V}, \mathsf{Sim})$ is a QA-ZK argument system in the CRS model for a set of witness-relations $\mathcal{R}_{\mathsf{pars}} = \{\mathcal{R}_{\mathtt{lpar}}\}_{\mathtt{lpar} \in \mathrm{Supp}(\mathcal{D}_{\mathsf{pars}})}$ with $\mathtt{lpar}$ sampled from a distribution $\mathcal{D}_{\mathsf{pars}}$ over associated parameter language $\mathcal{L}_{\mathsf{pars}}$, if the properties (i-iii) hold. Here, Pgen are the public parameter pars and $\mathsf{K}_{\mathsf{crs}}$ the $\mathtt{crs}$ generation algorithms, P is the prover, V is the verifier, and Sim is the simulator.

(i) Perfect Completeness. For any λ, $\mathsf{pars} \in \mathsf{Pgen}(1^\lambda)$, $\mathtt{lpar} \in \mathcal{D}_{\mathsf{pars}}$, and $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$,

$$\Pr\left[(\mathtt{crs}, \tau) \leftarrow \mathsf{K}_{\mathsf{crs}}(\mathtt{lpar}); \langle \mathsf{P}(\mathtt{w}), \mathsf{V}\rangle_{\mathtt{crs}}(\mathtt{x}) = 1\right] = 1 \ .$$

(ii) Zero-Knowledge. For any λ, $\mathsf{pars} \in \mathsf{Pgen}(1^\lambda)$, $\mathtt{lpar} \in \mathcal{D}_{\mathsf{pars}}$, for any computationally unbounded adversary $\mathcal{A}$, $2 \cdot |\varepsilon^{zk} - 1/2| \approx_\lambda 0$, where $\varepsilon^{zk} :=$

$$\Pr\left[(\mathtt{crs}, \tau) \leftarrow \mathsf{K}_{\mathsf{crs}}(\mathtt{lpar}); (\mathtt{x}, \mathtt{w}) \leftarrow \mathcal{A}(\mathtt{crs}); b \leftarrow_\$ \{0,1\} : \langle \mathsf{P}_b, \mathcal{A}\rangle_{\mathtt{crs}}(x) = 1 \quad\right] \ .$$

Where P_b terminates with $\perp$ if $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}_{\mathtt{lpar}}$. If $b = 0$, P_b represents $\mathsf{P}(\mathtt{w})$ and if $b = 1$, P_b represents $\mathsf{Sim}(\tau)$.

(iii) Computational Quasi-Adaptive Soundness. For any PPT $\mathcal{A}$ and for all $\mathtt{x}$ s.t. $\neg(\exists \mathtt{w} : \mathcal{R}_{\mathtt{lpar}}(\mathtt{x}, \mathtt{w}))$,

$$\Pr\left[(\mathsf{pars}, \mathtt{lpar}) \leftarrow \mathsf{Pgen}(1^\lambda); (\mathtt{crs}, \tau) \leftarrow \mathsf{K}_{\mathsf{crs}}(\mathtt{lpar}) : \langle \mathcal{A}, \mathsf{V}\rangle_{\mathtt{crs}}(\mathtt{x}) = 1 \quad\right] \approx_\lambda 0 \ .$$

BDH Assumption. We require the following knowledge assumption:

Assumption 1 (BDH**-Knowledge Assumption [ABLZ17])** *We say that* Pgen *is* BDH-KE secure *for $\mathcal{R}$ if for any λ, $(\mathcal{R}, \mathsf{aux}_{\mathcal{R}}) \in \mathrm{range}(\mathcal{R}(1^\lambda))$, and PPT adversary $\mathcal{A}$ there exists a PPT extractor $\mathsf{Ext}_{\mathcal{A}}^{\mathsf{BDH}}$, such that*

$$\Pr\begin{bmatrix} r \leftarrow_r \mathsf{RND}(\mathcal{A}); \\ ([\alpha_1]_1, [\alpha_2]_2 || a) \leftarrow (\mathcal{A}||\mathsf{Ext}_{\mathcal{A}}^{\mathsf{BDH}})(\mathcal{R}, \mathsf{aux}_{\mathcal{R}}; \omega_{\mathcal{A}}) : \\ [\alpha_1]_1[1]_2 = [1]_1[\alpha_2]_2 \wedge a \neq \alpha_1 \end{bmatrix} \approx_\lambda 0 \ .$$

Note that the BDH assumption can be considered as a simple case of the PKE assumptionof [DFGK14] (where $\mathcal{A}$ is given as an input the tuple $\{([x^i]_1, [x^i]_2)\}_{i=0}^n$ for some $n \geq 0$, and assumed that if $\mathcal{A}$ outputs $([\alpha]_1, [\alpha]_2)$ then she knows $(a_0, a_1, \ldots, a_n)$, such that $\alpha = \sum_{i=0}^n a_i x^i$.) as used in the case of asymmetric pairings in [DFGK14]. Thus, BDH can be seen as an asymmetric-pairing version of the original KoE assumption [Dam92].

3 A New Framework for TSPHFs

In this section, we present our revisited TSPHF framework. Conceptually, we start from the GL-TSPHFs construction in [BBC+15] and show how we can modify the framework such that all three hashing algorithms hash, projhash, and thash yield hash values in $\mathbb{G}_1$ instead of $\mathbb{G}_T$. This yields a more efficient and "lighter" version of TSPHFs which we call *lighter* TSPHF (L-TSPHF). Our framework is parametrized by a SPHF Σ which is required to be pairing-free, but it is then instantiated in source group $\mathbb{G}_\iota$, $\iota \in \{1, 2\}$, of a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \bar{e})$.

In general, let us define the language for the SPHF Σ that fits the generic framework in [BBC+13] as follows:

$$\mathcal{L}' = \left\{ \mathtt{x} \in \mathbb{G}_\iota^{1\times n} : \exists \mathtt{w} \in \mathbb{Z}_p^{1\times k};\ \mathtt{x} = \mathtt{w}[\mathbf{\Gamma}]_\iota \right\},$$

where $\mathbf{\Gamma} \in \mathbb{Z}_p^{k\times n}$ is the language parameter and a full rank matrix $(n > k)$. As with the TSPHF framework in [BBC+13], our framework provides the algorithms verhp and thash and we recall that the verhp algorithm checks well-formedness of the projection key hp and thash computes the hash value tH, without knowing neither the witness $\mathtt{w}$ nor the hashing key hk. We recall that the hashing key of Σ is a vector $\mathsf{hk} = \alpha \leftarrow_\$ \mathbb{Z}_p^n$, while the projection key is, for a word $\mathtt{x} = [\theta]_1$, $\mathsf{hp} = [\mathbf{\Gamma}(\mathtt{x})]_\iota \alpha \in \mathbb{G}_\iota^k$ (it represents hp_1 in the *L-TSPHF*) and note that *L-TSPHF*s in our framework are GL-style irrespective whether the underlying SPHF Σ is GL- or KV-style.

Now, we briefly outline our construction idea. The Pgen algorithm outputs an additional CRS $\mathtt{crs}' = ([\mathbf{b}]_{3-\iota}, [\mathbf{b}\mathbf{\Gamma}]_\iota)$ and its trapdoor $\tau = \mathbf{b}\mathbf{\Gamma}$ where $\mathbf{b} \leftarrow_\$ \mathbb{Z}_p^{n\times k}$. Here, $\mathbf{\Gamma}$ is the language parameter which we mask in the trapdoor τ with a vector $\mathbf{b}$. This is to guarantee that thash does not know $\mathbf{\Gamma}$ but only $\tau = \mathbf{b}\mathbf{\Gamma}$.[5] Now the idea is that for a

[5] We note that in the SPHF/TSPHF and their applications in zero-knowledge proofs, one wants to simulate a proof without knowing the trapdoor $\mathbf{\Gamma}$ of the base elements of the statement to be proven.

```
$\mathsf{Pgen}(1^\lambda, \mathcal{L})$
    - Generate $\mathbf{\Gamma} \in \mathbb{Z}_p^{k \times n}$;
    - Generate $\mathbf{b} \leftarrow_\$ \mathbb{Z}_p^{n \times k}$;
    - $\mathtt{lpar} := [\mathbf{\Gamma}]_\iota$; $\mathtt{crs}' := ([\mathbf{b}]_{3-\iota}, [\tau]_\iota)$;
    - $\tau := \mathbf{b\Gamma}$; $\mathtt{crs} := (\mathtt{lpar}, \mathtt{crs}')$;
    - return $(\tau, \mathtt{crs})$.
$\mathsf{hashkg}(\mathcal{L})$
    - return $\mathsf{hk} \in \mathbb{Z}_p^n \leftarrow \Sigma.\mathsf{hashkg}(\mathcal{L})$.
$\mathsf{projkg}(\mathsf{hk}, \mathtt{crs}, \mathtt{x})$
    - $\mathsf{hp}_1 \leftarrow \Sigma.\mathsf{projkg}(\mathsf{hk}, \mathcal{L})$;
    - $\mathsf{hp}_2 := [\tau\alpha]_\iota \in \mathbb{G}_\iota^n$;
    - return $\mathsf{hp} := (\mathsf{hp}_1, \mathsf{hp}_2) \in \mathbb{G}_\iota^{k+n}$.

$\mathsf{hash}(\mathsf{hk}, \mathtt{crs}, \mathtt{x})$
    - return $\mathsf{H} \leftarrow \Sigma.\mathsf{hash}(\mathsf{hk}, \mathtt{lpar}, \mathtt{x})$.
$\mathsf{projhash}(\mathsf{hp}, \mathtt{crs}, \mathtt{x}, \mathtt{w})$
    - $\mathsf{pH} \leftarrow \Sigma.\mathsf{projhash}(\mathsf{hp}, \mathtt{lpar}, \mathtt{x}, \mathtt{w})$;
    - return $\mathsf{pH}$.
$\mathsf{verhp}(\mathsf{hp}, \mathtt{crs}, \mathtt{x})$
    - if $[\mathbf{b}]_{3-\iota} \cdot \mathsf{hp}_1 = [1]_{3-\iota} \cdot \mathsf{hp}_2$ return $1$.
    - else return $0$.
$\mathsf{thash}(\omega, \tau, \mathsf{hp}, \mathtt{crs}, \mathtt{x})$
    - $\mathsf{hk} \leftarrow \mathsf{Ext}_\mathsf{Z}(\mathtt{crs}, \mathsf{hp}; \omega)$;
    - return $\mathsf{tH} \leftarrow \mathtt{x} \cdot \mathsf{hk}$.
```

Fig. 2. Full construction of L-TSPHF$[\Sigma]$.

hashing key $\mathsf{hk} := \alpha \leftarrow_\$ \mathbb{Z}_p^n$ our projection key, besides the projection key of the SPHF Σ, contains a second component $\mathsf{hp}_2 = [\tau\alpha]_\iota \in \mathbb{G}_\iota^n$ which is a representation of hk and $\mathtt{crs}'$ in $\mathbb{G}_\iota$ (this is similar to TSPHFs). Then, by using the knowldege assumption LTSPHF-KE, we know that there exists an extractor $\mathsf{Ext}_\mathcal{A}$ knowing the random coins of $\mathcal{A}$ (or the random coins of projhash) which returns a hashing key α that could have been used to compute hp. Finally, the thash algorithm can use this information to generate the trapdoor hash tH (cf. Lemma 1 for details and the precise use of the LTSPHF-KE assumption).

3.1 *Lighter-TSPHF* (L-TSPHF)

Now we present our L-TSPHF framework, which relies on the knowledge assumption LTSPHF-KE (cf. Assumption 2) and require that for any efficient malicious projection key creator Z, there exists an efficient extractor Ext_Z, s.t. if Z, by using the random coins ω as an input, generates a projection key hp then Ext_Z, given the same input and ω, outputs the hashing key hk corresponding to hp.

Definition 3. *A L-TSPHF$[\Sigma]$ for language $\mathcal{L}$ based upon SPHF Σ is defined by the following algorithms:*

-$\mathsf{Pgen}(1^\lambda, \mathcal{L})$**:** *Takes a security parameter λ and language $\mathcal{L}$. Choose the trapdoor of language parameter ($\mathbf{\Gamma} \leftarrow_\$ \mathbb{Z}_p^{k \times n}$). Chooses the trapdoor $\mathbf{b} \leftarrow_\$ \mathbb{Z}_p^{n \times k}$ such that $\tau := \mathbf{b\Gamma}$ is a diagonal matrix of size $n \times n$. Sets $\mathtt{lpar} = ([\mathbf{\Gamma}]_\iota)$ and $\mathtt{crs}' = ([\mathbf{b}]_{3-\iota}, [\tau]_\iota)$. It outputs $(\tau, \mathtt{crs} := (\mathtt{lpar}, \mathtt{crs}'))$.*

-$\mathsf{hashkg}(\mathcal{L})$**:** *Takes a language $\mathcal{L}$ and outputs a hashing key $\mathsf{hk} := \alpha \leftarrow_\$ \mathbb{Z}_p^n$ of Σ for the language $\mathcal{L}$, i.e., return $\mathsf{hk} \leftarrow \Sigma.\mathsf{hashkg}(\mathcal{L})$.*

-$\mathsf{projkg}(\mathsf{hk}, \mathtt{crs}, \mathtt{x})$**:** *Takes a hashing key hk, a CRS crs, and a word x and computes a projection key $\mathsf{hp} := (\mathsf{hp}_1, \mathsf{hp}_2) \in \mathbb{G}_\iota^{k+1}$, where $\mathsf{hp}_1 = [\mathbf{\Gamma}\alpha]_\iota \in \mathbb{G}_1^k$ is the projection key of Σ, i.e., $\mathsf{hp}_1 \leftarrow \Sigma.\mathsf{projkg}(\mathsf{hk}, \mathcal{L})$, $\mathsf{hp}_2 = [\tau\alpha]_\iota \in \mathbb{G}_\iota^n$ is a representation of hashing key hk and $\mathtt{crs}'$ in $\mathbb{G}_\iota$.*

-$\mathsf{hash}(\mathsf{hk}, \mathtt{x})$**:** *Takes a hashing key hk, and a word x and outputs a hash $\mathsf{H} = \mathtt{x} \cdot \alpha \in \mathbb{G}_\iota$, being the hash of Σ, i.e., $\mathsf{H} \leftarrow \Sigma.\mathsf{hash}(\mathsf{hk}, \mathtt{x})$.*

```
$\mathsf{Exp}^{\mathsf{zk}\text{-}b}(\mathcal{A}, \mathcal{L}, \lambda)$
$(\tau, \mathtt{crs}) \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$;
$\omega \leftarrow_\$ \mathsf{RND}(\mathsf{Z})$;
$(\mathtt{x}, \mathtt{w}, \mathsf{hp}, \mathsf{st}) \leftarrow \mathsf{Z}(\mathtt{crs}; \omega)$;
$\mathsf{hk} \leftarrow \mathsf{Ext}_\mathsf{Z}(\mathtt{crs}; \omega)$;
$b \leftarrow_\$ \{0, 1\}$;
$b' \leftarrow \mathcal{A}^{\mathsf{O}_b(\cdot,\cdot)}(\mathtt{crs}, \mathtt{x}, \mathtt{w}, \mathsf{hp}, \mathsf{st})$;
return $\mathsf{verhp}(\mathtt{crs}, \mathsf{hp}, \mathtt{x})$
        $\wedge\, b' = b$.
```

```
$\mathsf{O}_0(\mathtt{x}, \mathtt{w})$
if $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}_\mathcal{L} \vee \mathsf{verhp}(\mathtt{crs}, \mathsf{hp}, \mathtt{x}) = 0$
    return $\perp$.
else $\mathsf{pH} \leftarrow \mathsf{projhash}(\mathsf{hp}, \mathtt{crs}, \mathtt{x}, \mathtt{w})$
    return $\mathsf{pH}$.

$\mathsf{O}_1(\mathtt{x}, \mathtt{w})$
if $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}_\mathcal{L} \vee \mathsf{verhp}(\mathtt{crs}, \mathsf{hp}, \mathtt{x}) = 0$
    return $\perp$.
else $\mathsf{tH} \leftarrow \mathsf{thash}(\omega, \tau, \mathsf{hp}, \mathtt{crs}, \mathtt{x})$
    return $\mathsf{tH}$,
```

Fig. 3. Experiment $\mathsf{Exp}^{\mathsf{zk}\text{-}b}(\mathcal{A}, \mathcal{L}, \lambda)$.

- $\mathsf{projhash}(\mathsf{hp}, \mathtt{x}, \mathtt{w})$**:** *Takes a projection key* $\mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2)$*, a word* $\mathtt{x}$*, and a witness* $\mathtt{w}$ *for* $\mathtt{x} \in \mathcal{L}$ *and outputs a hash* $\mathsf{pH} = \mathtt{w} \cdot \mathsf{hp}_1 \in \mathbb{G}_\iota$*, being the projective hash of* Σ*, i.e.,* $\mathsf{pH} \leftarrow \Sigma.\mathsf{projhash}(\mathsf{hp}_1, \mathtt{x}, \mathtt{w})$.
- $\mathsf{verhp}(\mathsf{hp}, \mathtt{crs})$**.** *Takes projection key* hp *and CRS* $\mathtt{crs}$*, and outputs 1 if* hp *is a valid projection key, and 0 otherwise.*
- $\mathsf{thash}(\omega, \tau, \mathsf{hp}, \mathtt{crs}, \mathtt{x})$**:** *Takes random coins* ω *of* $\mathsf{projhash}$*, trapdoor* τ *and a projection key* hp*, the CRS* $\mathtt{crs}$ *and word* $\mathtt{x}$*, and by using an* Ext *(underling a knowledge assumption) extracts* $\mathsf{hk} = \alpha$ *and outputs* $\mathsf{tH} = \mathtt{x} \cdot \alpha \in \mathbb{G}_\iota$.

We present the L-TSPHF[Σ] construction in Fig. 2. *L-TSPHFs* must satisfy the properties correctness, zero-knowledge and computational smoothness. We note that the zero-knowledge property is called *soundness* in the context of TSPHFs in [BBC$^+$13] and was later called *zero-knowledge* in [Ben16]. We use the more intuitive term zero-knowledge, since in a typical application of (T)SPHFs, it guarantees that a malicious hp generator does not learn anything from seeing a projective hash pH (which depends on the witness) compared to when she seeing a (trapdoor) hash value H (which does not depend on the witness).

Perfect correctness. For any $(\tau, \mathtt{crs} = (\mathtt{lpar}, \mathtt{crs}')) \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$ and any word $\mathtt{x} \in \mathcal{L}$ with witness $\mathtt{w}$, for any $\mathsf{hk} \leftarrow \mathsf{hashkg}(\mathcal{L})$, any $\omega \leftarrow_\$ \mathsf{RND}(\mathsf{projkg})$ and for $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk}, \mathtt{crs}, \mathtt{x}; \omega)$, we have: $\mathsf{verhp}(\mathsf{hp}, \mathtt{crs}) = 1$, and $\mathsf{hash}(\mathsf{hk}, \mathtt{crs}, \mathtt{x}) = \mathsf{thash}(\omega, \tau, \mathsf{hp}, \mathtt{crs}, \mathtt{x})$.

Zero-Knowledge. There exist deterministic algorithms thash, verhp, s.t. the following holds. For any PPT algorithm Z, there exists a PPT extractor Ext_Z, s.t. for all λ, and unbounded $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{Z},\mathcal{A}}(\lambda) \approx_\lambda 0$, where

$$\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{Z},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{zk}\text{-}0}(\mathcal{A}, \mathcal{L}, \lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{zk}\text{-}1}(\mathcal{A}, \mathcal{L}, \lambda) = 1]|,$$

and the zero-knowledge experiment is defined in Fig. 3.

Computational Smoothness. Is based on that of TSPHFs and note that the trapdoors with exception of the one to $\mathtt{crs}_\mathcal{L} = \mathtt{lpar}$ are dropped and the full $\mathtt{crs}$ is given to the adversary. For a language $\mathcal{L}$ and adversary $\mathcal{A}$, the advantage is defined as follows:

$$\mathsf{Adv}^{\mathsf{csmooth}}_{\mathcal{L},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{csmooth}-0}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{csmooth}-1}(\mathcal{A}, \lambda) = 1]|.$$

The computational smoothness experiment is also defined in Fig. 4.

```
$\mathsf{Exp}^{\mathsf{csmooth\text{-}b}}(\mathcal{A}, \lambda)$

– $(\tau, \mathtt{crs}) \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L}), \mathsf{hk} \leftarrow \Sigma.\mathsf{hashkg}(\mathcal{L})$;
– $(\mathtt{x}, \mathsf{st}) \leftarrow \mathcal{A}(\mathtt{crs})$;
– $\mathsf{hp} \leftarrow \Sigma.\mathsf{projkg}(\mathsf{hk}, \mathtt{crs}, \mathtt{x})$;
– $b \leftarrow_\$ \{0, 1\}$;
– **if** $b = 0$, then $\mathsf{H} \leftarrow \Sigma.\mathsf{hash}(\mathsf{hk}, \mathtt{crs}, \mathtt{x})$, **else** $\mathsf{H} \leftarrow_\$ \Omega$;
– **return** $\mathcal{A}(\mathtt{crs}, \mathtt{x}, \mathsf{hp}, \mathsf{H}, \mathsf{st})$.
```

Fig. 4. Experiment $\mathsf{Exp}^{\mathsf{csmooth\text{-}b}}$ for computational smoothness.

New knowledge assumption. Let L-TSPHF$[\Sigma]$ be the *L-TSPHF*. To prove the ZK property of our construction, we need to rely on a new assumption we call LTSPHF-KE. Inspired by the knowledge assumption of [AKL21], we first define a new assumption and then prove its security under the HAK assumptions in Lemma 1. The knowledge assumption is to postulate that given a valid hp, one can efficiently extract $\mathsf{hk} = \alpha$. More precisely, the LTSPHF-KE assumption is the core of the ZK proof of the L-TSPHF$[\Sigma]$ construction in Theorem 1. There, we assume that if an adversary $\mathcal{A}$ outputs a hp accepted by verhp, then there exists an extractor $\mathsf{Ext}_\mathcal{A}$ that by knowing the secret coins of $\mathcal{A}$, returns $\mathsf{hk} = \alpha$ where hk was used to compute hp.

Assumption 2 (LTSPHF-KE) *Fix $n > k \geq 1$. Let L-TSPHF$[\Sigma]$ be the Lighter-TSPHF. The* LTSPHF-KE *assumption holds relative to* Pgen *for any PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathsf{Ext}_\mathcal{A}$, such that $\mathsf{Adv}^{\mathrm{hak}}_{\mathcal{A}}(\lambda) :=$*

$$\Pr\begin{bmatrix}\mathtt{crs} \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L}); \omega \leftarrow_\$ \mathsf{RND}(\mathcal{A}); \mathsf{hp} \leftarrow \mathcal{A}(\mathtt{crs}, \omega); \\ \mathsf{hk} \leftarrow \mathsf{Ext}_\mathcal{A}(\mathtt{crs}, \omega) : \mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2) \wedge \mathsf{verhp}(\mathsf{hp}, \mathtt{crs}) = 1 \wedge \mathsf{hp}_1 \neq \mathbf{\Gamma}\alpha.\end{bmatrix} \approx_\lambda 0.$$

We now show that LTSPHF-KE is secure under a hash-algebraic knowledge (HAK) assumption from [Lip19].

Lemma 1 (Security of LTSPHF-KE). *Fix $n > k \geq 1$. Then* LTSPHF-KE *holds relative to* Pgen *under the ϵ-HAK assumption.*

Due to the lack of space the proof of Lemma 1 is deferred to Appendix A.

Theorem 1. *The L-TSPHF$[\Sigma]$ in Fig. 2 is complete, if the* LTSPHF-KE *assumption holds, then it is zero-knowledge and if* DDH *holds in $\mathbb{G}_\iota$, $\iota \in \{1, 2\}$ then L-TSPHF$[\Sigma]$ has computational smoothness.*

Proof. **(i: Completeness)** This is straightforward from the construction.

(ii: Zero-knowledge) Let Z be a subverter that computes hp so as to break the zero-knowledge property. The subverter Z gets as an input the language parameter $\mathtt{crs}$ and a random tape ω, and outputs hp^* and some auxiliary state st. Let $\mathcal{A}$ be the adversary from Lemma 1. Note that $\mathsf{RND}(\mathcal{A}) = \mathsf{RND}(\mathsf{Z})$. Under the LTSPHF-KE assumption, there exists an extractor $\mathsf{Ext}_\mathcal{A}$, such that if $\mathsf{verhp}(\mathtt{crs}, \mathsf{hp}^*, \mathtt{x}) = 1$ then $\mathsf{Ext}_\mathcal{A}(\mathtt{crs}, \mathsf{hp}^*; \omega)$ outputs hk.

Fix $\mathtt{crs}$, $\omega \in \mathsf{RND}(\mathsf{Z})$, hp^* and run $\mathsf{Ext}_\mathsf{Z}(\mathtt{crs}, \mathsf{hp}^*; \omega)$ to obtain hk. It clearly suffices to show that if $\mathsf{verhp}(\mathtt{crs}, \mathsf{hp}^*, \mathtt{x}) = 1$ and $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}$ then

$$\begin{aligned}\mathsf{O}_0(\mathtt{x}, \mathtt{w}) &= \mathsf{projhash}(\mathsf{hp}^*, \mathtt{crs}, \mathtt{x}, \mathtt{w}) = \mathsf{pH} \quad , \\ \mathsf{O}_1(\mathtt{x}, \mathtt{w}) &= \mathsf{thash}(\omega, \tau, \mathsf{hp}^*, \mathtt{crs}) = \mathsf{tH}\end{aligned}$$

have the same distribution, where O_0 and O_1 work as in Fig. 3. This holds since from $\mathsf{verhp}(\mathtt{crs}, \mathsf{hp}, \mathtt{x}) = 1$ it follows $\mathsf{O}_0(\mathtt{x}, \mathtt{w}) = \mathsf{pH} = \mathsf{tH} = \mathsf{O}_1(\mathtt{x}, \mathtt{w})$. Hence, O_0 and O_1 have the same distribution.

(iii: Smoothness) The proof of smoothness is given in Appendix B. □

3.2 Comparison of the TSPHF Frameworks

In Table 1 we compare the efficiency of *L-TSPHF* with the GL-/KV-TSPHF constructions of [BBC+13] where $n > k$. Note that having $\mathbb{G}_1$ instead of $\mathbb{G}_T$ gives a factor ≥ 12 of bandwidth savings and also elements in $\mathbb{G}_2$ are typically twice the size of $\mathbb{G}_1$ for current type-III bilinear groups.

Table 1. Comparison between GL-/KV-TSPHF and *L-TSPHF*.

Scheme	$\lvert\mathsf{H}\rvert$	$\lvert\mathsf{hp}\rvert$
KV-TSPHF[BBC+13]	$\mathbb{G}_T$	$k \times \mathbb{G}_1 + n \times \mathbb{G}_2$
GL-TSPHF[BBC+13]	$\mathbb{G}_T$	$k \times \mathbb{G}_1 + n \times \mathbb{G}_2$
L-TSPHF$[\Sigma]$	$\mathbb{G}_1$	$(k + n) \times \mathbb{G}_1$

4 Updatable L-TSPHF

In this section, we propose an updatable version of *L-TSPHF*s (called *uL-TSPHF*s). The goal of updatability is to protect smoothness (analogous to soundness for zk-SNARKs in [GKM+18]) in the case the $\mathtt{crs}$ may be subverted, by requiring that at least one among the creator and all parties performing an update of $\mathtt{crs}$ is honest.

We define *uL-TSPHF*s by roughly following the definitional guidelines of [GKM+18] for updatable zk-SNARKs and [Lip20] for updatable QA-NIZKs. But in contrast to these ad-hoc constructions for particular instances of proof systems, our updatable L-TSPHF framework is generic and can be considered as a new cryptography tool with updatable ZK (cf. Section 5) being one application. Similar to QA-NIZKs, since the CRS of *uL-TSPHF*s depends on a language parameter $\mathbf{\Gamma}$, its security definitions are different when compared to zk-SNARKs. We redefine updatable versions of completeness, zero-knowledge and smoothness correspondingly. In order to satisfy the hiding property of the CRS updating procedure (following [Lip20] we call the CRS henceforth *key*), we add the requirement that an updated key and a fresh key are indistinguishable. Additionally we add key-updating and update-verification algorithms with the corresponding security requirements: key-update completeness, key-update hiding,

```
$\mathsf{Exp}^{\text{u-zk-b}}(\mathcal{A}, \mathcal{L}, \lambda)$
$(\tau, \mathtt{tc}, \mathtt{crs} = (\mathtt{lpar}, \mathtt{crs}')) \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$;
$\omega \leftarrow\!\$\ \mathsf{RND}(\mathsf{Z})$;
$(\mathsf{hp}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}'_{\mathsf{int}}, \mathsf{st}) \leftarrow \mathsf{Z}(\mathtt{crs}; \omega)$;
$\mathsf{hk} \leftarrow \mathsf{Ext}_{\mathsf{Z}}(\mathtt{crs}; \omega)$;
$b \leftarrow\!\$\ \{0, 1\}$;
$b' \leftarrow \mathcal{A}^{\mathsf{O}_b(\cdot,\cdot)}(\mathtt{crs}, \mathtt{crs}'_{\mathsf{upd}}, \mathsf{hp}, \mathsf{st})$;
return $\mathsf{UpdVer}(\mathtt{crs}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) = 1 \wedge$
$\mathsf{verhp}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathsf{hp}, \mathtt{x}) = 1 \wedge b' = b$.

$\mathsf{O}_0(\mathtt{x}, \mathtt{w})$
if $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}_{\mathcal{L}} \vee$
 $\mathsf{verhp}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathsf{hp}, \mathtt{x}) = 0$
  return $\perp$.
else
 $\mathsf{pH} \leftarrow \mathsf{projhash}(\mathsf{hp}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x}, \mathtt{w})$;
 return $\mathsf{pH}$.

$\mathsf{O}_1(\mathtt{x}, \mathtt{w})$
if $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}_{\mathcal{L}} \vee$
 $\mathsf{verhp}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathsf{hp}, \mathtt{x}) = 0$
  return $\perp$.
else
 $\mathsf{tH} \leftarrow \mathsf{thash}(\omega, \tau, \mathsf{hp}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x})$;
 return $\mathsf{tH}$.
```

Fig. 5. Experiment $\mathsf{Exp}^{\text{u-zk-b}}(\mathcal{A}, \mathcal{L}, \lambda)$.

strong key-update hiding, key-update smoothness, and key-update zero-knowledge.

uL-TSPHF. An updatable L-TSPHF (uL-TSPHF) has the following PPT algorithms in addition to $(\mathsf{Pgen}, \mathsf{hashkg}, \mathsf{projkg}, \mathsf{projhash}, \mathsf{hash}, \mathsf{verhp}, \mathsf{thash})$.

- **-**$\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}')$**.** Is a deterministic CRS verification algorithm which, given both $\mathtt{lpar}$ and $\mathtt{crs}'$, checks if they are well-formed.
- **-**$\mathsf{Upd}(\mathtt{lpar}, \mathtt{crs}')$**.** Is a randomized key updater algorithm, given $\mathtt{lpar}$, $\mathtt{crs}'$, generates a new updated $\mathtt{crs}'$ ($\mathtt{crs}'_{\mathsf{upd}}$), and returns $(\mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}, \mathtt{tc}_{\mathsf{upd}})$ where $\mathtt{tc}_{\mathsf{upd}}$ is some trapdoor of the updated CRS $\mathtt{crs}'_{\mathsf{upd}}$. $\mathtt{crs}_{\mathsf{int}}$ contains elements which intuitively can bee seen as a proof that updating is done correctly.
- **-**$\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}})$**.** Is a deterministic key updated verification algorithm which, given $\mathtt{crs}'$ and $\mathtt{crs}'_{\mathsf{upd}}$, and $\mathtt{crs}_{\mathsf{int}}$ checks correctness of the updating procedure.

Security Requirements. We note that all security notions are given for a single update, but they can be generalized for many updates by using standard hybrid arguments (cf. [GKM+18]).

Updatable key correctness. For any $(\tau, \mathtt{tc}, \mathtt{crs} = (\mathtt{lpar}, \mathtt{crs}')) \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$, $(\mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}, \mathtt{tc}_{\mathsf{upd}}) \leftarrow \mathsf{Upd}(\mathtt{lpar}, \mathtt{crs}')$, it holds that $\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) = 1$. In addition, if $\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) = 1$, then $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}') = 1$ iff $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}) = 1$.

Updatable key hiding. For any $(\tau, \mathtt{tc}, \mathtt{crs} = (\mathtt{lpar}, \mathtt{crs}')) \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$, $(\mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}, \mathtt{tc}_{\mathsf{upd}}) \leftarrow \mathsf{Upd}(\mathtt{lpar}, \mathtt{crs}')$, then we have: $(\mathtt{crs}', \mathtt{tc}) \approx_\lambda (\mathtt{crs}'_{\mathsf{upd}}, \mathtt{tc}_{\mathsf{upd}})$.

Updatable strong key hiding. The key-update hiding holds if one of the following holds:

- the original $\mathtt{crs}$ was honestly generated and the key-update verifies: $(\tau, \mathtt{tc}, \mathtt{crs} = (\mathtt{lpar}, \mathtt{crs}')) \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$, and $\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) = 1$.

```
$\mathsf{Exp}^{\text{F-ucsmooth-b}}(\mathcal{A}, \lambda)$
$(\tau, \mathtt{tc}, \mathtt{lpar}, \mathtt{crs}') \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$;
$\mathsf{hk} \leftarrow \Sigma.\mathsf{hashkg}(\mathcal{L})$;
$(\mathtt{x}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}, \mathsf{st}) \leftarrow \mathcal{A}(\mathtt{crs})$;
if $\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) \neq 1$
  return $\perp$.
$\mathsf{hp} \leftarrow \Sigma.\mathsf{projkg}(\mathsf{hk}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x})$;
$b \leftarrow_{\$} \{0, 1\}$;
if $b = 0$
  $\mathsf{H} \leftarrow \Sigma.\mathsf{hash}(\mathsf{hk}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x})$,
else $\mathsf{H} \leftarrow_{\$} \Omega$;
return $\mathcal{A}(\mathtt{crs}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x}, \mathsf{hp}, \mathsf{H}, \mathsf{st})$.
```

```
$\mathsf{Exp}^{\text{B-ucsmooth-b}}(\mathcal{A}, \lambda)$
$(\tau, \mathtt{lpar}) \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$;
$\mathsf{hk} \leftarrow \Sigma.\mathsf{hashkg}(\mathcal{L})$;
$(\mathtt{crs}', \mathsf{st}) \leftarrow \mathcal{A}(\mathtt{lpar})$;
if $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}') \neq 1$
  return $\perp$.
$(\mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}, \mathtt{tc}_{\mathsf{upd}}) \leftarrow \mathsf{Upd}(\mathtt{lpar}, \mathtt{crs}')$;
$(\mathtt{x}, \mathsf{st}) \leftarrow \mathcal{A}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathsf{st})$;
$\mathsf{hp} \leftarrow \Sigma.\mathsf{projkg}(\mathsf{hk}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x})$;
$b \leftarrow_{\$} \{0, 1\}$;
if $b = 0$
  $\mathsf{H} \leftarrow \Sigma.\mathsf{hash}(\mathsf{hk}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x})$;
else $\mathsf{H} \leftarrow_{\$} \Omega$;
return $\mathcal{A}(\mathtt{crs}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x}, \mathsf{hp}, \mathsf{H}, \mathsf{st})$.
```

Fig. 6. Experiments $\mathsf{Exp}^{\text{x-ucsmooth-b}}$ with $\mathsf{x} \in \{\mathsf{F}, \mathsf{B}\}$ for updatable computational smoothness.

– the original $\mathtt{crs}$ verifies and the key-update was honest: $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}') = 1$, and $(\mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}, \mathtt{tc}_{\mathsf{upd}}) \leftarrow \mathsf{Upd}(\mathtt{lpar}, \mathtt{crs}')$.

Updatable Completeness. For any $(\tau, \mathtt{tc}, \mathtt{crs} = (\mathtt{lpar}, \mathtt{crs}')) \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$, any $(\mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}, \mathtt{tc}_{\mathsf{upd}}) \leftarrow \mathsf{Upd}(\mathtt{lpar}, \mathtt{crs}')$ and any word $\mathtt{x} \in \mathcal{L}$ with witness $\mathtt{w}$, for any $\mathsf{hk} \leftarrow \mathsf{hashkg}(\mathcal{L})$, any $\omega \leftarrow_{\$} \mathsf{RND}(\mathsf{projkg})$ and $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk}, \mathtt{crs}, \mathtt{x}; \omega)$, we have: $\mathsf{verhp}(\mathsf{hp}, \mathtt{crs}, \mathtt{x}) = 1$, $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}) = 1$, and $\mathsf{hash}(\mathsf{hk}, \mathtt{crs}, \mathtt{x}) = \mathsf{thash}(\omega, \tau, \mathsf{hp}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x})$.

Updatable zero-knowledge. There exist deterministic algorithms thash, verhp, s.t. the following holds. For any PPT subverter Z, there exists a PPT extractor $\mathsf{Ext}_{\mathsf{Z}}$, s.t. for all λ, and unbounded $\mathcal{A}$, $\mathsf{Adv}^{\text{u-zk}}_{\mathsf{Z},\mathcal{A}}(\lambda) \approx_\lambda 0$, where

$$\mathsf{Adv}^{\text{u-zk}}_{\mathsf{Z},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\text{u-zk-0}}(\mathcal{A}, \mathcal{L}, \lambda) = 1] - \Pr[\mathsf{Exp}^{\text{u-zk-1}}(\mathcal{A}, \mathcal{L}, \lambda) = 1]|.$$

and the updatable zero-knowledge experiment is defined in Fig. 5.

Updatable computational smoothness. It holds iff both, the updatable forward computational smoothness, and the updatable backward computational smoothness as shown in Fig. 6 hold. For a language $\mathcal{L}$ and adversary $\mathcal{A}$, the advantage is defined as follows:

$$\mathsf{Adv}^{\text{ucsmooth}}_{\mathcal{L},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\text{ucsmooth-0}}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Exp}^{\text{ucsmooth-1}}(\mathcal{A}, \lambda) = 1]|.$$

Subsequently, we show that updatable smoothness and updatable zero-knowledge follow from simpler security requirements. This means that it will suffice to prove computational smoothness, zero-knowledge, completeness, updatable key correctness and updatable strong key hiding. The dependencies between the security properties are summarized as follows:

Updatable completeness. It suffices to prove updatable key correctness and completeness.

Updatable zero-knowledge. It suffices to prove updatable key correctness and the extractability of $\mathtt{tc}_{\mathsf{upd}}$, and zero-knowledge.

Updatable computational smoothness. It suffices to prove updatable key correctness, computational smoothness, and updatable strong key hiding. We prove the above statements in the following Lemmas 2 to 4.

Lemma 2. *Assume uL-TSPHF$[\Sigma]$ is updatable key correct and complete. Then uL-TSPHF$[\Sigma]$ has updatable completeness.*

Lemma 3. *Assume uL-TSPHF$[\Sigma]$ is updatable key correct, the trapdoor* $\mathtt{tc}_{\mathsf{upd}}$ *extractable and zero-knowledge. Then uL-TSPHF$[\Sigma,]$ is updatable zero-knowledge.*

Lemma 4. *Assume uL-TSPHF$[\Sigma]$ is computational smooth and updatable strongly key hiding. Then (i) uL-TSPHF$[\Sigma]$ is updatable forward computational smooth. (ii) If uL-TSPHF$[\Sigma]$ is additionally updatable key correct, then uL-TSPHF$[\Sigma]$ is updatable backward computational smooth.*

The proofs of these lemmas are straightforward and provided in Appendix C.1, Appendix C.2 and Appendix C.3.

Lemma 5. *The uL-TSPHF$[\Sigma]$ in Fig. 7 is (i) updatable key correct. Then assuming $\mathbf{b} \leftarrow_{\$} \mathcal{D}_B$, where the distribution $\mathcal{D}_B$ satisfies the condition that for independent random variables $\beta_i \leftarrow_{\$} \mathcal{D}_B$, for $i \in \{1, 2\}$, we have $\beta_1 + \beta_2 \leftarrow_{\$} \mathcal{D}_B$; Then the construction in Fig. 7 is (ii) updatable key hiding, (iii) updatable strong key hiding.*

Due to the lack of space we defer the proof to Appendix C.4.

Theorem 2. *The uL-TSPHF$[\Sigma]$ in Fig. 7 has* updatable completeness*, if the* LTSPHF-KE *and* BDH *assumptions hold, it is* statistically updatable zero-knowledge*, and if* DDH *holds in $\mathbb{G}_\iota$, $\iota \in \{1, 2\}$ then it has* updatable forward computational smoothness. *Assuming that the preconditions of Lemma 5 are satisfied, then the it has* updatable backward computational smoothness.

Proof.

(i: Statistically updatable completeness.) The proof follows from Lemma 5 (*uL-TSPHF$[\Sigma]$* is updatable key correct), Theorem 1 (*uL-TSPHF$[\Sigma]$* is complete), and Lemma 2 (updatable completeness follows from updatable key correctness and completeness).

(i: Statistically updatable zero-knowledge.) The proof follows from Lemma 5 (*uL-TSPHF$[\Sigma]$* is updatable key correct), Theorem 1 (*L-TSPHF$[\Sigma]$* is zero-knowledge), $\mathtt{tc}_{\mathsf{upd}}$ extractability (if $\mathsf{UpdVer}(.) = 1$, more precisely $[\mathbf{b}^*]_1[1]_2 = [1]_1[\mathbf{b}^*]_2$, then under BDH assumption, there exists an extractor $\mathsf{Ext}_{\mathsf{Z}}^{\mathsf{BDH}}$, given random coin ω_{Z}, outputs $\mathtt{tc}_{\mathsf{upd}} = \mathbf{b}^*$), and Lemma 3 (updatable zero-knowledge follows from updatable key correctness, the $\mathtt{tc}_{\mathsf{upd}}$ extractability and zero-knowledge).

(iii: Updatable computational smoothness.) This follows from Theorem 1 (*uL-TSPHF$[\Sigma]$* is computationally smooth under the DDH assumption), and Lemma 5 (any *uL-TSPHF$[\Sigma]$* is updatable strongly key hiding), and Lemma 4 (any computational smooth and updatable strongly key hiding *uL-TSPHF$[\Sigma]$* is also updatable computational smooth). $\square$

```
$\mathsf{Pgen}(1^\lambda, \mathcal{L})$
  - Generate $\mathbf{\Gamma} \in \mathbb{Z}_p^{k \times n}$;
  - Generate $\mathbf{b} \leftarrow_\$ \mathbb{Z}_p^{n \times k}$; such that $\tau := \mathbf{b}\mathbf{\Gamma}$ e a diagonal matrix of size $n \times n$.
  - $\mathtt{lpar} := ([\mathbf{\Gamma}]_\iota)$; $\mathtt{crs}' := ([\mathbf{b}]_{3-\iota}, [\tau]_\iota)$;
  - return $(\tau, \mathtt{tc} =: \mathbf{b}, \mathtt{crs} = (\mathtt{lpar}, \mathtt{crs}'))$.
$\mathsf{Upd}(\mathtt{lpar}, \mathtt{crs}')$
  - $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}')$ : if $[\tau]_\iota \cdot [1]_{3-\iota} \neq [\mathbf{b}]_{3-\iota} \cdot [\mathbf{\Gamma}]_\iota$ return $0$; else return $1$.
  - $\mathsf{Upd}(\mathtt{lpar}, \mathtt{crs}')$ : $\mathbf{b}^* \leftarrow_\$ \mathbb{Z}_p^{n \times k}$, $\mathtt{crs}_{\mathsf{int}} := (\tau^* := [\mathbf{b}^*\mathbf{\Gamma}]_\iota, [\mathbf{b}^*]_1, [\mathbf{b}^*]_2)$;
  - return $\mathtt{crs}'_{\mathsf{upd}} := ([\tau_{\mathsf{upd}} := \tau + \tau^*]_\iota, [\mathbf{b}_{\mathsf{upd}} := \mathbf{b} + \mathbf{b}^*]_{3-\iota}), \mathtt{crs}_{\mathsf{int}}, \mathtt{tc}_{\mathsf{upd}} := \mathbf{b}^*$.
$\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}_{\mathsf{int}}, \mathtt{crs}'_{\mathsf{upd}})$
  - $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}})$ : $[\tau^*]_\iota \cdot [1]_{3-\iota} \overset{?}{=} [\mathbf{b}^*]_{3-\iota} \cdot [\mathbf{\Gamma}]_\iota$
  - $\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}_{\mathsf{int}}, \mathtt{crs}'_{\mathsf{upd}})$ : $[\mathbf{b}_{\mathsf{upd}}]_{3-\iota} \overset{?}{=} [\mathbf{b}]_{3-\iota} + [\mathbf{b}^*]_{3-\iota} \wedge [\mathbf{b}^*]_1 \cdot [1]_2 = [1]_1 \cdot [\mathbf{b}^*]_2$
        $\wedge [\tau_{\mathsf{upd}}]_\iota \overset{?}{=} [\tau]_\iota + [\tau^*]_\iota \wedge [\tau_{\mathsf{upd}}]_\iota \cdot [1]_{3-\iota} \overset{?}{=} [\mathbf{b}]_{3-\iota} \cdot [\mathbf{\Gamma}]_\iota + [\mathbf{b}^*]_{3-\iota} \cdot [\mathbf{\Gamma}]_\iota$;
  - if $\mathsf{crsVer}(.) = 1 \wedge \mathsf{UpdVer}(.) = 1$ return $1$; else return $0$.
$\mathsf{hashkg}(\mathcal{L})$
  - $\mathsf{hk} \in \mathbb{Z}_p^n \leftarrow \Sigma.\mathsf{hashkg}(\mathcal{L})$;
$\mathsf{projkg}(\mathsf{hk}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x})$
  - $\mathsf{hp}_1 \leftarrow \Sigma.\mathsf{projkg}(\mathsf{hk}, \mathcal{L})$; $\mathsf{hp}_2 := [\tau_{\mathsf{upd}}\alpha]_\iota \in \mathbb{G}_\iota^n$;
  - return $\mathsf{hp} := (\mathsf{hp}_1, \mathsf{hp}_2) \in \mathbb{G}_\iota^{k+n}$.
$\mathsf{hash}(\mathsf{hk}, \mathtt{x})$
  - return $\mathsf{H} \leftarrow \Sigma.\mathsf{hash}(\mathsf{hk}, \mathtt{x})$.
$\mathsf{projhash}(\mathsf{hp}, \mathtt{x}, \mathtt{w})$
  - return $\mathsf{pH} \leftarrow \Sigma.\mathsf{projhash}(\mathsf{hp}, \mathtt{x}, \mathtt{w})$.
$\mathsf{verhp}(\mathsf{hp}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}})$
  - $\mathsf{verhp}(\mathsf{hp}, \mathtt{crs})$ : if $[\mathbf{b}_{\mathsf{upd}}]_{3-\iota} \cdot [\mathbf{\Gamma}\alpha]_\iota = [\tau_{\mathsf{upd}}\alpha]_\iota [1]_{3-\iota}$ return $1$; else return $0$.
$\mathsf{thash}(\omega, \tau, \mathsf{hp}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}})$
  - $\mathtt{tc}_{\mathsf{upd}} = \mathbf{b}^* \leftarrow \mathsf{Ext}_{\mathsf{Z}}^{\mathsf{BDH}}(\mathtt{lpar}, \mathtt{crs}'; \omega_{\mathsf{Z}})$;
  - By using $\mathsf{Ext}^{\mathsf{LTSPHF\text{-}KE}}$, extract $\mathsf{hk} = \alpha$ and then computes $\mathsf{tH} := \mathtt{x} \cdot \mathsf{hk}$.
```

Fig. 7. Full construction of updatable *L-TSPHF* (*uL-TSPHF*).

Concrete Construction of Updatable L-TSPHF. Finally, in Fig. 7 we present the full construction of *uL-TSPHF*s. Intuitively, since $\mathtt{crs}'$ consists of (bracketed) matrices, we can construct an updating process where all $\mathtt{crs}'$ elements are updated additively. We remark that the subverter Z could be the updater and $\mathcal{A}$ could be the malicious projection key generator and note that we can have $\mathtt{crs}'_{\mathsf{upd}} = \mathtt{crs}'$.

5 Applications of (Updatable) *L-TSPHF*s

In this section we discuss the application *uL-TSPHF*s to updatable ZK arguments. Due to the lack of space we defer applications of *L-TSPHF*s to ZK arguments to Appendix D.1 (and an efficient ZK argument for correct encryption of a valid Waters signature [Wat05] to Appendix D.2). In Appendix F we discuss the applications to updatable Password-Authenticated Key-Exchange (uPAKE).

Updatable Zero-Knowledge Arguments. We now construct a generic framework for updatable QA-ZK Arguments from updatable L-TSPHFs. The generic framework is depicted in Fig. 8. Before we analyze the security of the updatable ZK argument, we

```
CRS Generation
    - Run $\mathsf{Pgen}(1^\lambda, \mathcal{L})$ algorithm of uL-TSPHF and return $(\tau := \mathbf{b\Gamma}, \mathtt{tc} := \mathbf{b}, \mathtt{crs} := (\mathtt{lpar}, \mathtt{crs}'))$.
CRS Update $\mathsf{Upd}(\mathtt{lpar}, \mathtt{crs}')$
    - Run Upd algorithm of uL-TSPHF and return $(\mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}, \mathtt{tc}_{\mathsf{upd}})$;
Verify Update $\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}_{\mathsf{int}}, \mathtt{crs}'_{\mathsf{upd}})$
    - Run UpdVer algorithm of uL-TSPHF and if $\mathsf{crsVer}(.) = 1 \wedge \mathsf{UpdVer}(.) = 1$ return $1$; else return $0$.
Verifier($\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathcal{L}, \mathtt{x}$)
    - $\mathsf{hk} = \alpha \in \mathbb{Z}_p^n \leftarrow \Sigma.\mathsf{hashkg}(\mathcal{L})$; $\mathsf{H} \in \mathbb{G}_\iota \leftarrow \Sigma.\mathsf{hash}(\alpha, \mathtt{lpar}, \mathtt{x})$;
    - $\mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2) \leftarrow \Sigma.\mathsf{projkg}(\alpha, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x})$;
    Send hp to prover.
Prover($\mathsf{hp}, \mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathcal{L}, \mathtt{x}, \mathtt{w}$)
    - $\mathsf{pH} \in \mathbb{G}_\iota \leftarrow \Sigma.\mathsf{projhash}(\mathsf{hp}_1, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x}, \mathtt{w})$;
    - if $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}) = 1 \wedge \mathsf{verhp}(\mathsf{hp}_1, \mathsf{hp}_2, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}) = 1$ return $\pi := \mathsf{pH}$; else return $\perp$.
Verification($\mathsf{H}, \pi, \mathtt{x}$)
    - if $\mathsf{H} = \pi$ return accept; else return reject.
Simulator($\omega, \tau, \mathsf{hp}, \mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathcal{L}, \mathtt{x}$)
    - $\mathsf{tH} \leftarrow \mathsf{thash}(\omega, \tau, \mathsf{hp}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}})$;
    - if $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}) = 1 \wedge \mathsf{verhp}(\mathsf{hp}_1, \mathsf{hp}_2, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}) = 1$ return $\pi := \mathsf{tH}$; else return $\perp$.
```

Fig. 8. Updatable ZK Argument from L-TSPHF.

present the new definitions for updatable forward and backward soundness.

Updatable forward soundness. for any $\mathtt{lpar} \in \mathrm{im}(\mathsf{Pgen}(1^\lambda))$, PPT $\mathcal{A}$ and for all $\mathtt{x}$ s.t. $\neg(\exists \mathtt{w} : \mathcal{R}_{\mathtt{lpar}}(\mathtt{x}, \mathtt{w}))$,

$$\Pr\left[\begin{matrix}(\mathtt{crs}, \mathtt{tc}) \leftarrow \mathsf{K}_{\mathsf{crs}}(\mathtt{lpar}); (\mathtt{crs}_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) \leftarrow \mathcal{A}(\mathtt{lpar}, \mathtt{crs}) : \\ \mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}, \mathtt{crs}_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) = 1 \quad \wedge \langle \mathcal{A}, \mathsf{V} \rangle_{\mathtt{crs}}(\mathtt{x}) = 1\end{matrix}\right] \approx_\lambda 0 \ .$$

Updatable backward soundness. for any $\mathtt{lpar} \in \mathrm{im}(\mathsf{Pgen}(1^\lambda))$, PPT $\mathcal{A}$ and for all $\mathtt{x}$ s.t. $\neg(\exists \mathtt{w} : \mathcal{R}_{\mathtt{lpar}}(\mathtt{x}, \mathtt{w}))$,

$$\Pr\left[\begin{matrix}\mathtt{crs} \leftarrow \mathcal{A}(\mathtt{lpar}); (\mathtt{crs}_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}, \mathtt{tc}_{\mathsf{upd}}) \leftarrow \mathsf{Upd}(\mathtt{lpar}, \mathtt{crs}) : \\ \mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}, \mathtt{crs}_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) = 1 \quad \wedge \langle \mathcal{A}, \mathsf{V} \rangle_{\mathtt{crs}}(\mathtt{x}) = 1\end{matrix}\right] \approx_\lambda 0 \ .$$

Updatable soundness. It holds iff both, the updatable forward soundness, and the updatable backward soundness hold.

Theorem 3. *Let the uL-TSPHF be updatable key correct, updatable key hiding, updatable strong key hiding, statistically updatable zero-knowledge, and updatable computationally smooth. Then the updatable ZK argument in Fig. 8 is (i) updatable key correct, (ii) updatable key hiding, (iii) updatable strong key hiding, (iv) updatable complete, (v) updatable zero-knowledge, and (vi) updatable sound.*

The proof is straightforward and can be found in Appendix E.

Acknowledgements. This work received funding from European Union's Horizon 2020 ECSEL Joint Undertaking project under grant agreement n° 783119 (SECREDAS), from the European Union's Horizon 2020 research and innovation programme under grant agreement n°871473 (KRAKEN) and by the Austrian Science Fund (FWF) and netidee SCIENCE under grant agreement P31621-N38 (PROFET).

References

ABL$^+$19. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, Janno Siim, and Michal Zajac. UC-secure CRS generation for SNARKs. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 99–117. Springer, Heidelberg, July 2019.

ABLZ17. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2017.

ABP15. Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Disjunctions for hash proof systems: New constructions and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 69–100. Springer, Heidelberg, April 2015.

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

AJOR18. Masayuki Abe, Charanjit S. Jutla, Miyako Ohkubo, and Arnab Roy. Improved (almost) tightly-secure simulation-sound QA-NIZK with applications. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 627–656. Springer, Heidelberg, December 2018.

AKL21. Behzad Abdolmaleki, Hamidreza Khoshakhlagh, and Helger Lipmaa. Smooth zero-knowledge hash functions. Cryptology ePrint Archive, Report 2021/653, 2021. `https://eprint.iacr.org/2021/653`.

ALSZ20. Behzad Abdolmaleki, Helger Lipmaa, Janno Siim, and Michal Zajac. On QA-NIZK in the BPK model. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 590–620. Springer, Heidelberg, May 2020.

ARS20. Behzad Abdolmaleki, Sebastian Ramacher, and Daniel Slamanig. Lift-and-shift: Obtaining simulation extractable subversion and updatable SNARKs generically. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20*, pages 1987–2005. ACM Press, November 2020.

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

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

BBC$^+$15. Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. Cryptology ePrint Archive, Report 2015/188, 2015. `http://eprint.iacr.org/2015/188`.

Ben16. Fabrice Ben Hamouda--Guichoux. *Diverse modules and zero-knowledge*. PhD thesis, École Normale Supérieure, Paris, France, 2016.

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*, volume 6571 of *LNCS*, pages 403–422. Springer, Heidelberg, March 2011.

BFS16. Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, December 2016.

BGG17. Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. Cryptology ePrint Archive, Report 2017/602, 2017. https://eprint.iacr.org/2017/602.

BMP00. Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, Heidelberg, May 2000.

Boy09. Xavier Boyen. HPAKE: Password authentication secure against cross-site user impersonation. In Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors, *CANS 09*, volume 5888 of *LNCS*, pages 279–298. Springer, Heidelberg, December 2009.

BP13. Fabrice Benhamouda and David Pointcheval. Trapdoor smooth projective hash functions. Cryptology ePrint Archive, Report 2013/341, 2013. http://eprint.iacr.org/2013/341.

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

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

BSCG$^+$15. Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE, 2015.

CFQ19. Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019.

CHM$^+$20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

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

Dam92. Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 445–456. Springer, Heidelberg, August 1992.

DFGK14. George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. In Palash Sarkar and Tetsu

Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 532–550. Springer, Heidelberg, December 2014.

DRZ20. Vanesa Daza, Carla Ràfols, and Alexandros Zacharakis. Updateable inner product argument with logarithmic verifier and applications. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 527–557. Springer, Heidelberg, May 2020.

EHK$^+$13. Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Heidelberg, August 2013.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

Fuc18. Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg, March 2018.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

GKM$^+$18. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-snarks. In *Annual International Cryptology Conference*, pages 698–728. Springer, 2018.

GMR89. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

GOS06. Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg, May / June 2006.

Gro10. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

Had10. Satoshi Hada. Secure obfuscation for encrypted signatures. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 92–112. Springer, Heidelberg, May / June 2010.

HK98. Shai Halevi and Hugo Krawczyk. Public-key cryptography and password protocols. In Li Gong and Michael K. Reiter, editors, *ACM CCS 98*, pages 122–131. ACM Press, November 1998.

JR12. Charanjit S. Jutla and Arnab Roy. Relatively-sound NIZKs and password-based key-exchange. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 485–503. Springer, Heidelberg, May 2012.

JR13. Charanjit S. Jutla and Arnab Roy. Shorter quasi-adaptive NIZK proofs for linear subspaces. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2013.

JR14. Charanjit S. Jutla and Arnab Roy. Switching lemma for bilinear tests and constant-size NIZK proofs for linear subspaces. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 295–312. Springer, Heidelberg, August 2014.

JR15. Charanjit S. Jutla and Arnab Roy. Dual-system simulation-soundness with applications to UC-PAKE and more. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 630–655. Springer, Heidelberg, November / December 2015.

Kil92. Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

Kle14. Achim Klenke. *Probability Theory: A Comprehensive Course*. Springer, 2014.

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

KW15. Eike Kiltz and Hoeteck Wee. Quasi-adaptive NIZK for linear subspaces revisited. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 101–128. Springer, Heidelberg, April 2015.

Lip12. Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.

Lip19. Helger Lipmaa. Simulation-extractable snarks revisited. Cryptology ePrint Archive, Report 2019/612, 2019. `https://eprint.iacr.org/2019/612`.

Lip20. Helger Lipmaa. Key-and-argument-updatable QA-NIZKs. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 645–669. Springer, Heidelberg, September 2020.

LPJY14. Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Non-malleability from malleability: Simulation-sound quasi-adaptive NIZK proofs and CCA2-secure encryption from homomorphic signatures. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 514–532. Springer, Heidelberg, May 2014.

LPJY15. Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Compactly hiding linear spans - tightly secure constant-size simulation-sound QA-NIZK proofs and applications. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 681–707. Springer, Heidelberg, November / December 2015.

Mac01. Philip D. MacKenzie. More efficient password-authenticated key exchange. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 361–377. Springer, Heidelberg, April 2001.

MBKM19. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

Wat05. Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, Heidelberg, May 2005.

Appendix

A Proof of Lemma 1

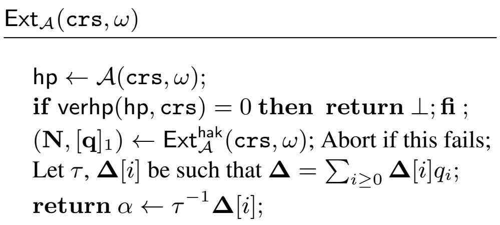
The proof is inspired by that of the KWKE assumption in [ALSZ20]. Let $\mathcal{A}$ is a LTSPHF-KE adversary that, given the CRS $\mathtt{crs} = ([\mathbf{\Gamma}]_1, \mathtt{crs}')$ and randomness $\omega \leftarrow_\$ \mathsf{RND}(\mathcal{A})$ as input, outputs and hp, s.t. with probability $\epsilon_\mathcal{A}$, $\mathsf{verhp}(\mathsf{hp}, \mathtt{crs}) = 1$. Denote $\mathbf{\Delta} := \tau\alpha^\top \in \mathbb{Z}_p^{n\times 1}$ where $\tau = \mathbf{b\Gamma}$. Let $\mathsf{verhp}(\mathsf{hp}, \mathtt{crs}) = 1$, i.e., $[\mathbf{b}]_2[\mathsf{hp}_1]_1 = [1]_2[\mathbf{\Delta}]_1 \in \mathbb{G}_T^{n\times 1}$. Let $\mathsf{Ext}_\mathcal{A}^{\mathsf{hak}}$ be the extractor, existence of which is guaranteed by the ε-HAK assumption. Fig. 9 depicts the extractor $\mathsf{Ext}_\mathcal{A}$, where $[q_i]_1$ for $i > 0$ are group elements created by $\mathcal{A}$ (for which she does not know the discrete logarithm) in $\mathbb{G}_1$, and $q_0 = 1$. Due to the HAK assumption, $\mathsf{Ext}_\mathcal{A}^{\mathsf{hak}}$ can extract $\mathbf{N}$ and $[\mathbf{q}]_1$, such that

$$\begin{bmatrix} \mathsf{vect}(\mathbf{\Delta}) \\ \mathsf{hp}_1 \end{bmatrix}_1 = \mathbf{N} \begin{bmatrix} 1 \\ \mathbf{q} \end{bmatrix}_1 \in \mathbb{G}_1^{n+k} \enspace .$$

Here, $\mathrm{vect}(B)$ denotes the vectorization of a matrix B. Thus, e.g., $\tau_{ij} = \sum_{t\geq 0}^{|\mathbf{q}_2|+1} N_{k(i-1)+j,t}\mathbf{q}_t$. Given $\mathbf{N}$, one can efficiently compute matrices $\mathsf{hp}_1[i]$ and $\mathbf{\Delta}[i]$, s.t. the polynomials

$$\mathsf{hp}_1(\mathbf{Q}) := \textstyle\sum_{j\geq 0} \mathsf{hp}_1[j]Q_j \in \mathbb{Z}_p^{1\times k}[\mathbf{Q}] \enspace , \mathbf{\Delta}(\mathbf{Q}) := \textstyle\sum_{i\geq 0} \mathbf{\Delta}[i]Q_i \in \mathbb{Z}_p^{n\times 1}[\mathbf{Q}] \enspace .$$

satisfy $[\mathsf{hp}_1]_1 = [\mathsf{hp}_1(\mathbf{q})]_1$, and $[\mathbf{\Delta}]_1 = [\mathbf{\Delta}(\mathbf{q})]_1$.



```
$\mathsf{Ext}_\mathcal{A}(\mathtt{crs}, \omega)$
    $\mathsf{hp} \leftarrow \mathcal{A}(\mathtt{crs}, \omega)$;
    if $\mathsf{verhp}(\mathsf{hp}, \mathtt{crs}) = 0$ then return $\perp$; fi ;
    $(\mathbf{N}, [\mathbf{q}]_1) \leftarrow \mathsf{Ext}_\mathcal{A}^{\mathsf{hak}}(\mathtt{crs}, \omega)$; Abort if this fails;
    Let $\tau$, $\mathbf{\Delta}[i]$ be such that $\mathbf{\Delta} = \sum_{i\geq 0} \mathbf{\Delta}[i]q_i$;
    return $\alpha \leftarrow \tau^{-1}\mathbf{\Delta}[i]$;
```

Fig. 9. Extractors $\mathsf{Ext}_\mathcal{A}(\mathtt{crs}, \omega)$ in the proof of Lemma 1

Assume that $\mathcal{A}(\mathtt{crs}; \omega)$ was successful. We execute $\mathsf{Ext}_\mathcal{A}(\mathtt{crs}; \omega)$ and obtain either α or $\perp$. From the fact that $[\mathbf{b}]_2[\mathsf{hp}_1]_1 = [1]_2[\mathbf{\Delta}]_1 \in \mathbb{G}_T^{n\times 1}$, the verification polynomial

$$W(\mathbf{Q}) := \mathbf{b}\left(\textstyle\sum_{i\geq 0} \mathsf{hp}_1[i]Q_i\right) \in \mathbb{Z}_p^{n\times 1} - \left(\textstyle\sum_{i\geq 0} \mathbf{\Delta}[i]Q_i\right) \in \mathbb{Z}_p^{n\times 1}$$

satisfies $W(\mathbf{q}) = \mathbf{0}$. Following the blueprint of security proofs in the HAK framework [Lip19], we consider the following two cases, $W(\mathbf{Q}) = \mathbf{0}$ as a polynomial and $W(\mathbf{Q}) \neq \mathbf{0}$ but $W(\mathbf{q}) = \mathbf{0}$.

Case 1: $\mathrm{vect}(W(\mathbf{Q})) = \mathbf{0}_{1\times nk}$ *as a polynomial.* Since Q_j is indeterminate for all $i > 0$, the coefficients W_i of Q_i of $\mathrm{vect}(W(\mathbf{Q})) = \sum_{i\geq 0} W_iQ_i$ must be equal to $\mathbf{0}_{1\times nk}$ for all $i \geq 0$. In particular,

$$\mathbf{b} \cdot \mathsf{hp}_\mathbf{1}[i] = \mathbf{\Delta}[i] \in \mathbb{Z}_p^{n\times 1} \quad , \quad i \geq 0 \enspace . \tag{1}$$

```
$\mathsf{Exp}^{\mathsf{int}}_{\mathcal{B}}(1^\lambda, \mathcal{L})$
$[\mathbf{\Gamma}]_1 \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$;
$\mathbf{b} \leftarrow_\$ \mathbb{Z}_p^{1\times k}$;
Fix $[\mathbf{x}]_1 \in \mathbb{G}_1^n$;
$b \leftarrow_\$ \{0,1\}$;
$b' \leftarrow \mathcal{B}^{\mathsf{O}_b(\cdot,\cdot)}([\mathbf{\Gamma}]_1, [\mathbf{x}]_1, \mathbf{b})$;
return $b = b'$.
```

```
$\mathsf{O}_0([\mathbf{\Gamma}]_1, [\mathbf{x}]_1)$
$\alpha \leftarrow_\$ \mathbb{Z}_p^n$;
For $i \in [1,k], j \in [1,n]$ :
return $([\Gamma_{ij}\alpha_j]_1; [x_j\alpha_j]_1)$.

$\mathsf{O}_1([\mathbf{\Gamma}]_1, [\mathbf{x}]_1)$
$\alpha \leftarrow_\$ \mathbb{Z}_p^n$; $u_i \leftarrow_\$ \mathbb{Z}_p$
For $i \in [1,k], j \in [1,n]$ :
return $([\Gamma_{ij}\alpha_j]_1; [u_j]_1)$.
```

Fig. 10. Experiment $\mathsf{Exp}^{\mathsf{int}}_{\mathcal{B}}(1^\lambda, \mathcal{L})$ for the proof of smoothness in Theorem 1

Since τ is invertible (as it is honestly chosen). Define $\alpha(\mathbf{Q}) := \tau^{-1}\mathbf{\Delta}\mathbf{Q} = \tau^{-1}\sum_{i\geq 0}\mathbf{\Delta}[i]Q_i \in \mathbb{Z}_p^{n\times 1}[\mathbf{Q}]$. Let $\alpha := \alpha(\mathbf{y})$. Since $\tau(\mathbf{y})$ is invertible then from $\mathbf{\Delta}[i] = \sum_i \tau y_i \alpha[i] = \mathbf{b}\sum_i \mathbf{\Gamma}\alpha[i] = \mathbf{b}\mathsf{hp}_1$. Finally, define $\alpha := \tau^{-1}\mathbf{\Delta}[i_0] \in \mathbb{Z}_p^n$. Note that so defined α can be extracted since we know the coefficients of $\tau(\mathbf{Q})$ and $\mathbf{\Delta}(\mathbf{Q})$. Clearly, α is a valid hk since $\mathbf{\Gamma}\alpha = \tau^{-1}[i_0]\mathbf{\Delta}[i_0]\mathbf{\Gamma} = \sum \mathsf{hp}[i]Q_i = \mathsf{hp}(\mathbf{Q})$.

Case 2: $\mathrm{vect}(W(\mathbf{Q})) \neq \mathbf{0}$ *but* $\mathrm{vect}(W(\mathbf{q})) = \mathbf{0}$. Next, following [Lip19], we consider separately the "non-hashing" case ($\mathcal{A}$ creates no random elements $[q_{\iota i}]_\iota$) and the "hashing" case $\mathcal{A}$ creates at least one random element that has high min-entropy). In the non-hashing case, the verification polynomial is equal to the integer matrix $W := \mathbf{b}[0]\cdot\mathsf{hp}_1[0] - \mathbf{\Delta}[0] \in \mathbb{Z}_p^{n\times 1}$. Recall that $\mathrm{vect}(W(\mathbf{Q})) \neq \mathbf{0}$ but $\mathrm{vect}(W(\mathbf{q})) = \mathbf{0}$. Since there are no created group elements, this is clearly impossible (the polynomial $\mathrm{vect}(W)$ is constant, and we need $\mathrm{vect}(W) = 0$ and $\mathrm{vect}(W) \neq 0$ at the same time). Thus, $\mathcal{A}$ cannot succeed in the non-hashing case.

Consider now the "hashing" case when $\mathcal{A}$ has created at least one high min-entropy group element q_k (say, in $\mathbb{G}_1$). Clearly, $\mathrm{vect}(W(\mathbf{Q}))$ is a degree-1 polynomial in any indeterminate Q_i. Thus, by the high min-entropy version of the Schwartz-Zippel lemma, since $H_\infty([q_s]_1) = \omega(\log\lambda)$, then the probability $2^{-\sum_s H_\infty([q_s]_1)}$ that $\mathrm{vect}(W(\mathbf{q})) = 0$ is negligible. Hence, the probability that $\mathcal{A}$, who created at least one (high min-entropy) group element $[q_k]_1$, can make the verifier accept is negligible. $\square$

B Proof of Smoothness of Theorem 1

The proof is similar to the smoothness proof of [BBC+13] but with some modifications. We first reduce the smoothness to the following computational assumption: for all λ and PPT adversary $\mathcal{B}$, $|\mathsf{Exp}^{\mathsf{int}}_{\mathcal{B}}(1^\lambda, \mathcal{L}) - 1/2| \approx_\lambda 0$, where $\mathsf{Exp}^{\mathsf{int}}_{\mathcal{B}}(1^\lambda, \mathcal{L})$ is depicted in Fig. 10. Let adversary $\mathcal{B}$ be allowed to make only one query to the oracle O_b to obtain a tuple $([\Gamma_{ij}\alpha_j]_1, [z_j]_1)$ where $z_j = x\alpha_j$ or $z_j = u_j$.

Let $\mathcal{A}$ be the adversary against computational smoothness. We now construct the following adversary $\mathcal{B}$ against the *intermediate assumption*.

The adversary $\mathcal{B}$ works as follows:

- $[\mathbf{\Gamma}]_1 \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$; $\mathbf{b} \leftarrow_\$ \mathbb{Z}_p^{1\times k}$; $\mathtt{crs} = ([\mathbf{\Gamma}, \mathbf{b}\mathbf{\Gamma}]_1, [\mathbf{b}]_2)$;
- $\mathtt{x} \leftarrow \mathcal{A}(\mathtt{crs})$;
- $([\Gamma_{ij}\alpha_j]_1, [z_j]_1) \leftarrow \mathsf{O}_b([\mathbf{\Gamma}]_1, \mathtt{x})$, where if $b = 1$, $z_j = \mathtt{x}_j\alpha_j$ for $i = \{1,\dots,k\}$, $j = \{1,\dots,n\}$. Otherwise $z_j \leftarrow_\$ \mathbb{Z}_p$;

- $\mathsf{hp}_1 \leftarrow (\sum_{j=1}^{n} \mathbf{\Gamma}_{1j}\alpha_j, \ldots, \sum_{j=1}^{n} \mathbf{\Gamma}_{kj}\alpha_j) = [\mathbf{\Gamma}\alpha]_1$; $\mathsf{hp}_2 \leftarrow \mathbf{b}[\mathbf{\Gamma}\alpha]_1$;
- $\mathsf{H} \leftarrow \sum_{j=1}^{j=n}[z_j]_1$;
- $b_{\mathcal{A}} \leftarrow \mathcal{A}(\mathtt{crs}, \mathtt{x}, (\mathsf{hp}_1, \mathsf{hp}_2), \mathsf{H})$;
- $\mathbf{return}\ b' \leftarrow b_{\mathcal{A}}$.

Thus, $\mathcal{A}$ is successful in breaking the soundness game iff $\mathcal{B}$ is successful in breaking the *intermediate assumption*.

We now show that the *intermediate assumption* can be reduced to the DDH problem, i.e., it is hard to distinguish the two distributions, $\{[1, a, b, ab]_1\}$ and $\{[1, u, b, ab]_1\}$ where $a, b, u \leftarrow_{\$} \mathbb{Z}_p$. Let $\mathcal{D}$ be the adversary against this problem, such that given $T = \{[1, c, b, ab]_1\}$, it outputs 1 if $c = a$ and 0 otherwise.

Given the tuple T, $\mathcal{D}$ uses $\mathcal{B}$ as a subroutine. In particular, given a DDH tuple, $\mathcal{D}$ generates kn DDH tuples $[u_i, v_i, w_i]_1$, for $i \in [1 \mathinner{..} n]$, by random self-reducibility of DDH and sets $[\mathbf{u}]_1 = ([u_1]_1, \ldots, [u_n]_1) \in \mathbb{G}^n$, $[\mathbf{v}]_1 = ([v_{ij}]_1)_{i\in[k], j\in[n]} \in \mathbb{G}^{k\times n}$, and $[\mathbf{w}]_1 = ([w_1]_1, \ldots, [w_n]_1) \in \mathbb{G}^n$. Then she feeds $\mathcal{B}$ with $[\mathbf{u}]_1$. Also $\mathcal{D}$ plays the role of the challenger for in the experiment $\mathsf{Exp}_{\mathcal{B}}^{int}(1^\lambda, \mathcal{L})$ in Fig. 10 and when $\mathcal{B}([\mathbf{u}]_1)$ calls the oracle O_b, the adversary $\mathcal{D}$, first chooses $\alpha \leftarrow_{\$} \mathbb{Z}_p^n$ and answers with $([v_{ij}w_j]_1, [c_j]_1)$ where $c_j = x_j w_j$ or $z_j = u_j$. Then $\mathcal{D}$ returns $\mathcal{B}$'s output. Thus, $\mathcal{D}$ is successful in breaking the DDH problem iff $\mathcal{B}$ is successful in breaking the *intermediate assumption*. $\square$

C Omitted Proofs of Updatable L-TSPHF

C.1 Proof of Lemma 2

From updatable key correctness, we have $\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) = 1$. Then, by the completeness property, $\mathsf{verhp}(\mathtt{lpar}, \mathtt{crs}', \mathsf{hp}, \mathtt{x}) = 1$ and $\mathsf{hash}(\mathsf{hk}, \mathtt{crs}, \mathtt{x}) = \mathsf{thash}(\omega, \tau, \mathsf{hp}, \mathtt{crs}, \mathtt{x})$. $\square$

C.2 Proof of Lemma 3

Due to updatable key correctness, we have $\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) = 1$ and $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}) = 1$. Then, by $\mathtt{tc}_{\mathsf{upd}}$ extractable property (if $\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) = 1$, there exist an extractor $\mathsf{Ext}_{\mathsf{Z}}$ that extracts $\mathtt{tc}_{\mathsf{up}}$), and the zero-knowledge property, $\mathsf{verhp}(\mathtt{lpar}, \mathtt{crs}', \mathsf{hp}, \mathtt{x}) = 1$ and there exist an extractor $\mathsf{Ext}_{\mathsf{Z}}$ that extracts hk such that $\mathsf{thash}(\tau, \mathsf{hp}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}) = \mathsf{projhash}(\mathsf{hp}, \mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}, \mathtt{x}, \mathtt{w})$. $\square$

C.3 Proof of Lemma 4

(i: Updatable forward computational smoothness) By updatable strong key hiding, $\mathtt{crs}'_{\mathsf{upd}}$ comes from the correct distribution. Thus, by computational smoothness, it is computationally hard to distinguish the valid hash H with randomly chosen H.

(ii: Updatable backward computational smoothness) From updatable key correctness it follows that in the definition of forward updatable computational smoothness, we can replace the condition $\mathsf{UpdVer}(\mathtt{lpar}, \mathtt{crs}', \mathtt{crs}'_{\mathsf{upd}}, \mathtt{crs}_{\mathsf{int}}) = 1$ with the condition $\mathsf{crsVer}(\mathtt{lpar}, \mathtt{crs}'_{\mathsf{upd}}) = 1$. Because of the updatable strong key hiding, $\mathtt{crs}'_{\mathsf{upd}}$ is indistinguishable from an honestly generated $\mathtt{crs}'_{\mathsf{upd}}$. From computational smoothness, we now obtain updatable backward computational smoothness. □

C.4 Proof of Lemma 5

Before we present the proof, we recall a definition from [Kle14].

Definition 4 (14.46 in [Kle14]). *Let $I \in (0,1)$ be a semigroup. A family $u = \{u_t : t \in I\}$ of probability distributions on $\mathcal{R}^d$ is called a convolution semigroup if $u_{l+t} = u_l * u_t$ holds for all $l, t \in I$.*

Proof. **(i: Updatable key correctness.)** It is straightforward from the verifying updating phase in Fig. 7.

(ii: Updatable key hiding.) Let if $\beta_i \leftarrow_{\$} \mathcal{D}_B$ for $i \in \{1,2\}$, then $\beta_1 + \beta_2 \leftarrow_{\$} \mathcal{D}_B$. Then, $\mathcal{D}_B^{*2} = \mathcal{D}_B$, where $\mathcal{D}^{*l}$ is the l-th convolution power of $\mathcal{D}$. That is, $\mathcal{D}_B$ is a stable distribution. Note that one can generalize that requirement when one allows scaling factors. For example, when we define $\mathbf{b}_{\mathsf{upd}} \leftarrow \mathbf{b} + \mathbf{b}^*$, one can consider distributions of $\mathcal{D}_B$ of matrices where each matrix entry is either uniformly random or a constant. For the proof of key-update hiding, we need a convolution semigroup (cf. Definition 4) consisting of a single element. However, if we apply scaling factors, we can use more general convolution semigroups. Thus, since $\mathtt{crs}'$ is honestly created, $\mathbf{tau}$, $\mathbf{b}$ and so, $\mathtt{crs}'_{\mathsf{upd}}$ contains $\tau + \tau^* = (\mathbf{b} + \mathbf{b}^*)\mathbf{\Gamma} = \tau_{\mathsf{upd}}$. Due to the assumption on $\mathcal{D}_B$, $\mathtt{crs}'$ and $\mathtt{crs}'_{\mathsf{upd}}$ come from the same distribution.

(iii: Updatable strong key hiding.) We need to prove updatable key hiding for two different cases: (1) when $\mathtt{crs}$ was created honestly, and (2) when $\mathtt{crs}_{\mathsf{upd}}$ was honestly updated. For the case (1), since $\mathtt{crs}$ is honestly created, it contains $([\mathbf{\Gamma}, \tau]_\iota, [\mathbf{b}]_{3-\iota})$. Since $\mathsf{UpdVer}(.) = 1$, $\mathbf{b}_{\mathsf{upd}} = \mathbf{b} + \mathbf{b}^*$, thus by the assumption on $\mathcal{D}_B$, $\mathbf{b}_{\mathsf{upd}}$ comes from the correct distribution, and $\tau_{\mathsf{upd}} = \mathbf{b}_{\mathsf{upd}}\mathbf{\Gamma} = \tau + \tau^*$. This holds under the assumption that $\mathcal{D}_B$ is an ideal of a convolution semigroup. For the case (2), since $\mathtt{crs}$ verifies, thus by $\mathsf{crsVer}(.) = 1$, we have $[\tau]_\iota[1]_{3-\iota} = [\mathbf{b}]_{3-\iota}[\mathbf{\Gamma}]_\iota$. Since $\mathtt{crs}_{\mathsf{upd}}$ was honestly updated: for correctly distributed $\mathbf{b}^*$, $\mathbf{b}_{\mathsf{upd}} = \mathbf{b} + \mathbf{b}^*$, and $\tau_{\mathsf{upd}} = \tau + \tau^*$, and thus $[\tau_{\mathsf{upd}}]_\iota[1]_{3-\iota} = [\mathbf{b}_{\mathsf{upd}}]_{3-\iota}[\mathbf{\Gamma}]_\iota$. Because $\mathbf{b}^*$ has the correct distribution, also $\mathbf{b}_{\mathsf{upd}}\mathbf{\Gamma}$ has the correct distribution. □

D Zero-Knowledge Arguments from L-TSPHFs

In this section we construct a generic framework for 2-round ZK arguments (ZKAs) from *L-TSPHF*. In Supplementary Material D.2, we show as a concrete application how to construct an efficient ZK argument to prove the correct encryption of a valid Waters signature [Wat05].

D.1 ZK Arguments from *L-TSPHF*

In this section we construct a generic framework for quasi-adaptive zero-knowledge arguments from *L-TSPHF* for any linear language. The generic framework is depicted in Fig. 11.

Theorem 4. *Let L-TSPHF*$[\Sigma]$ *be complete, zero-knowledge, and computational smooth. Then the ZK argument in Fig. 11 is (i) complete, (ii) zero-knowledge, and (iii) sound.*

```
CRS Generation
    - Run $\mathsf{Pgen}(1^\lambda, \mathcal{L})$ algorithm of L-TSPHF$[\Sigma]$ and return $(\tau, \mathtt{crs} = (\mathtt{lpar}, \mathtt{crs}'))$.
Verifier($\mathtt{crs}, \mathcal{L}, \mathtt{x}$)
    $\mathsf{projkg}(\mathsf{hk}, \mathtt{crs}, \mathtt{x})$ :
    - $\mathsf{hk} := \alpha \in \mathbb{Z}_p^n \leftarrow \Sigma.\mathsf{hashkg}(\mathcal{L})$; $\mathsf{H} \leftarrow \Sigma.\mathsf{hash}(\alpha, \mathtt{lpar}, \mathtt{x})$;
    - $\mathsf{hp} \leftarrow \Sigma.\mathsf{projkg}(\alpha, \mathtt{lpar}, \mathtt{crs}', \mathtt{x})$;
    Send hp to prover.
Prover($\mathsf{hp}, \mathtt{crs}, \mathcal{L}, \mathtt{x}, \mathtt{w}$)
    - if $\mathsf{verhp}(\mathsf{hp}, \mathtt{crs}) = 0$ return $\perp$;
    - $\mathsf{pH} \in \mathbb{G}_\iota \leftarrow \Sigma.\mathsf{projhash}(\mathsf{hp}, \mathtt{lpar}, \mathtt{x}, \mathtt{w})$;
    - if $\mathsf{verhp}(.) = 1$ then return $\pi := \mathsf{pH}$; else return $\perp$.
Simulator($\omega, \tau, \mathsf{hp}, \mathtt{crs}, \mathcal{L}, \mathtt{x}$)
    - if $\mathsf{verhp}(\mathsf{hp}, \mathtt{crs}) = 0$ return $\perp$;
    - $\mathsf{tH} \leftarrow \mathsf{thash}(\omega, \tau, \mathsf{hp}, \mathtt{crs}, \mathtt{x})$;
    - if $\mathsf{verhp}(.) = 1$ then return $\pi := \mathsf{tH}$; else return $\perp$.
Verification($\mathsf{H}, \pi$)
    - if $\mathsf{H} = \pi$ then return accept; else return reject.
```

Fig. 11. Generic framework for 2-round ZK arguments from *L-TSPHF*$[\Sigma]$.

Proof.

(i: Completeness) It follows directly from completeness of *L-TSPHF*$[\Sigma]$ construction in Theorem 1.

(ii: Zero-knowledge) It follows from Theorem 1 (*L-TSPHF*$[\Sigma]$ is zero-knowledge), but the simulator first runs an additional algorithm verhp as it shown in Fig. 11.

(iii: Soundness) The proof for soundness follows from Theorem 1 (*L-TSPHF*$[\Sigma]$ is computationally smooth). $\square$

On Removing Interaction. We note that similar to the QA-NIZK construction in [ALSZ20], we can define our ZK arguments in Fig. 11 in a variant of the bare public-key (BPK) model , which is significantly weaker than the CRS model and arguably being the weakest public key or parameter based trust model. We use the variant of the BPK model in [ALSZ20], where only the verifier needs to have a public key (hp) and the key authority executes the functionality of an immutable bulletin board by storing the received public keys, But the CRS corresponding to the language parameter (the public key of the prover) is computed by a trusted party. Thus, one obtains designated-verifier QA-NIZK (DV-QA-NIZK) in the aforementioned variant of the BPK model from our ZK arguments in Fig. 11.

Efficiency Comparison. In Table 2, we compare the efficiency of ZK arguments based on *L-TSPHF* with the ZK arguments of [ALSZ20] where $n > k$. Let m be a security parameter in [KW15, ALSZ20] where $m = 1$ represents the optimized construction. In order to be consistent with [KW15, ALSZ20], we assume $\iota = 1$ which means that the proof π is in $\mathbb{G}_1$. We note that in [ALSZ20] we have $\mathtt{crs} = \mathtt{lpar}$ which only contains

Table 2. Comparison between the ZKAs of [ALSZ20] and ZKAs based on *L-TSPHF*.

Scheme	$\lvert\mathsf{crs}\rvert$	$\lvert\mathsf{comm}\rvert$	$\lvert\mathsf{proof}\rvert$
ZK argument [ALSZ20]	$kn \times \mathbb{G}_1$	$km \times \mathbb{G}_1 + (m^2 + nm) \times \mathbb{G}_2$	$m \times \mathbb{G}_1$
$\mathrm{ZK}_{L-TSPHF}$	$(k+1)n \times \mathbb{G}_1 + k \times \mathbb{G}_2$	$(k+1) \times \mathbb{G}_1$	$\mathbb{G}_1$

the language parameter. In our ZK construction from *L-TSPHF* the $\mathtt{crs}$ has $(n + k)$ additional elements, but we have significantly reduced communication overhead.

D.2 Verifiable Encryption of Waters Signatures

Now we demonstrate a concrete instantiation of our construction of ZK arguments from *L-TSPHF* in Fig. 11 for the language of ciphertexts of valid Waters signatures [Wat05] in an asymmetric bilinear group. These constructions can be used, for example, in optimistic fair exchanges of signatures [ASW98], obfuscation [Had10] or oblivious signature-based envelopes [BPV12]. We first describe linear encryption of Waters signatures and the corresponding language, and then show how we can obtain ZK arguments from *L-TSPHF* construction. The Waters signature scheme for asymmetric bilinear groups has been proposed and proven secure in [BFPV11] and it is defined below.

Waters Signatures. The Waters signature scheme for asymmetric bilinear group, has been proposed and proved in [BFPV11]. Briefly it is defined with the Four algorithms $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Versign})$ and works as follows,

-$\mathsf{Setup}(1^\lambda)$**:** Chooses a random vector $[\mathbf{u}]_1 \leftarrow_\$ \mathbb{G}_1^{k+1}$ which uses in the Waters hash function $[\mathcal{F}(M)]_1 = [u_0]_1 + \sum_{i=1}^{k} m_i [u_i]_1$ for $\mathbf{m} \in \{0,1\}^k$. Choose an extra generator $[v]_1 \leftarrow_\$ \mathbb{G}_1$ and sets $\mathsf{pars} = ([\mathbf{u}]_1, [v]_1)$;

-$\mathsf{KGen}(\mathsf{pars})$**:** Chooses $z \leftarrow_\$ \mathbb{Z}_p$ and computes the public verification key $\mathsf{vk} = ([z]_1, [z]_2)$ and its secret key $\mathsf{sk} = z[v]_1$.

-$\mathsf{Sign}(\mathsf{sk}, \mathbf{m}, s)$ **:** For a random $s \leftarrow_\$ \mathbb{Z}_p$ returns $\sigma = ([\sigma_1]_1, [\sigma_2]_1)$, where $[\sigma_1]_1 = \mathsf{sk} + s[\mathcal{F}(M)]_1$ and $[\sigma_2]_2 = ([s]_1, [s]_2)$;

$\mathsf{Versign}(\mathsf{vk}, \mathbf{m}, \sigma)$**:** First checks whether $[\sigma_{21}]_1[1]_2 = [1]_1[\sigma_{22}]_2$, then verifies $[\sigma_1]_1[1]_2 = [v]_1[z]_2 + [\mathcal{F}(M)]_1[\sigma_{22}]_2$.

The ungforgeability of this scheme is base on a variant of CDH assumption, given a set $([1]_1, [1]_2, [a]_1, [a]_2, [b]_1)$ for random $a, b \leftarrow_\$ \mathbb{Z}_p$, computing $[ab]_1$ is hard.

Linear Encryption of Waters signatures. Let $[v]_1 \leftarrow_\$ \mathbb{G}_1$ and $[\mathbf{u}]_1 \leftarrow_\$ \mathbb{G}_1^{k+1}$ which defines the Waters hash of a message $\mathbf{m} \in \{0,1\}^k$ as $[\mathcal{F}(M)]_1 = [u_0]_1 + \sum_{i=1}^{k} m_i[u_i]_1$. The verification key is $\mathsf{vk} = ([z]_1, [z]_2)$ and the associated signing key is $\mathsf{sk} = z[v]_1$ where $z \leftarrow_\$ \mathbb{Z}_p$. The signature on a message $\mathbf{m}$ is $\sigma = ([\sigma_1]_1, [\sigma_2]_1)$, where $[\sigma_1]_1 = \mathsf{sk} + s[\mathcal{F}(M)]_1$ and $[\sigma_2]_2 = ([s]_1, [s]_2)$ for some random $s \leftarrow_\$ \mathbb{Z}_p$. It can be verified by the following checking $[\sigma_{21}]_1[1]_2 = [1]_1[\sigma_{22}]_2$, and $[\sigma_1]_1[1]_2 = [v]_1[z]_2 + [\mathcal{F}(M)]_1[\sigma_{22}]_2$. The linear encryption public key is $\mathsf{pk} = ([t_1]_1, [t_2]_1)$ and the secret key is $\mathsf{sk} = (y_1, y_2) \leftarrow_\$ \mathbb{Z}_p^2$. The ciphertext of a Waters signature $\sigma = ([\sigma_1]_1, [\sigma_2]_1)$ is $\mathsf{c}_\sigma = (r_1[t_1]_1, r_2[t_2]_1, [r_1 + r_2]_1 + [\sigma_1]_1, [\sigma_{21}]_1, \mathsf{vk}_1)$ where $r_1, r_2 \leftarrow_\$ \mathbb{Z}_p^2$. Let us now formally show the resulting language $\mathcal{L}_\sigma$ is,

$$\left\{ \begin{matrix} \mathsf{c}_\sigma \in \mathbb{G}_1^5 : \exists (r_1, r_2, s, z) \in \mathbb{Z}_p^4;\ \mathsf{c}_1 = r_1[t_1]_1, \mathsf{c}_2 = r_2[t_2]_1, \\ \mathsf{c}_3 = [r_1 + r_2 + \sigma_1]_1, \mathsf{c}_4 = [s]_1, \mathsf{c}_5 = [z]_1 \end{matrix} \right\}.$$

For any $\mathsf{c}_\sigma = (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3.\mathsf{c}_4, \mathsf{c}_5)$ we have $[\boldsymbol{\Gamma}]_1 = \begin{pmatrix} [t_1]_1 & 1 & [1]_1 & 1 & 1 \\ 1 & [t_2]_1 & [1]_1 & 1 & 1 \\ 1 & 1 & [\mathcal{F}(M)]_1 & [1]_1 & 1 \\ 1 & 1 & [v]_1 & 1 & [1]_1 \end{pmatrix}$, and the witness $\mathtt{w} = (r_1, r_2, s, z)$ such that $\mathsf{c}_\sigma = \mathtt{w}[\boldsymbol{\Gamma}]_1$. Therefore, we can use the ZK arguments in Fig. 11.

E Proof of Theorem 3

(i,ii,iii: Updatable key correctness, hiding, strong hiding.) It follows directly from Lemma 5 (updatable L-TSPHF[Σ] is updatable key correct, updatable key hiding, updatable strong key hiding.

(iv,v: Updatable completeness, updatable zero-knowledge.) It follows from Theorem 2 (updatable L-TSPHF[Σ] is updatable complete and updatable zero-knowledge.

(vi: Updatable soundness.) The proof for updatable forward and backward soundness follow from Theorem 2 (updatable L-TSPHF[Σ] is updatable forward and backward smooth. □

F Updatable PAKE from Updatable L-TSPHFs

Now we show how the new primitive updatable *L-TSPHF* (*uL-TSPHF*) in Fig. 7 enables us to construct updatable two-round Password-Authenticated Key-Exchange (uPAKE), which allows us to reduce the trust in the setup. More precisely, we use our *uL-TSPHF* in Benhamouda *et al.*'s TSPHF-based PAKE framework [BBC$^+$13] and construct an updatable version of it. We recall that in Benhamouda *et al.*'s construction, the users need to trust a trusted third party to generate the CRS. Now, in our updatable PAKE the need of such trust is removed by using *uL-TSPHF* instead of *TSPHF* [BBC$^+$13].

Briefly in a PAKE protocol, each user U owns a word $\mathtt{x}_1$ in a certain language $\mathcal{L}_1$ and expects the other user to own a word $\mathtt{x}_2$ in a language $\mathcal{L}_2$. If everything is compatible (i.e., the languages are the expected languages and the words are indeed in the respective languages), the users compute a common high-entropy secret key, otherwise they learn nothing about the other user's values. In addition, we assume the two users have initially agreed on a common public language parameter $\mathtt{lpar}$ for the languages, but then they secretly parametrize the languages with the private parts $\mathtt{w}$: $\mathcal{L}_{lpar,\mathtt{w}}$ is the language they want to use. In addition, each user owns a word $\mathtt{x}$ in his language. We thus have to use *uL-TSPHF* on ciphertexts on $\mathtt{x}$ and $\mathtt{w}$, with a common value $\mathtt{lpar}$.

We note that Benhamouda *et al.*'s PAKE construction is secure in the Universal Composability (UC) framework, but due to the use of knowledge assumptions (non-falsifiable assumptions) in the prove of the zero-knowledge property of *uL-TSPHF*, achieving UC security for our updatable PAKE is unclear.

Basically most of the current non-UC PAKE protocols are in random oracle model and need at least two rounds [HK98, BPR00, BMP00, Mac01, Boy09]. In context of UC-secure protocols, there has been a sequence of papers [KV11, JR12, BBC+13, JR15] constructing one-round UC-secure PAKE protocol in the CRS model (using bilinear pairings). Benhamouda et.al [BBC+13] proposed the first CRS based UC-secure one-round PAKE protocol, but in a game-based model built on the BPR model [BPR00].

Following the security definitions of (non UC-secure) PAKE [BMP00], in general PAKE protocols must satisfy: (i) *completeness* meaning that for any real world adversary that faithfully passes messages between two user instances with complimentary roles and identities, both user instances accept, and (ii) *simulatability* meaning that for every efficient real world adversary $\mathcal{A}$, there exists an simulator Sim such that *RealWorld*($\mathcal{A}$) and *IdealWorld*(Sim) are computationally indistinguishable, where *RealWorld*($\mathcal{A}$) and *IdealWorld*(Sim) denote the transcript of $\mathcal{A}$'s and Sim's operations respectively.

Theorem 5. *Let uL-TSPHF in Fig. 7 be a updatable L-TSPHF, then the updatable PAKE protocol in Fig. 12 is complete and simulatable.*

Proof. **(i: Completeness)** This is straightforward from the construction and follows the directly from Lemma 5 (updatable L-TSPHF[Σ] is updatable key correct).

(ii: Simulatability) It follows directly from Lemma 5 (updatable L-TSPHF[Σ] is updatable key correct, updatable key hiding, updatable strong key hiding) and Theorem 2 (updatable L-TSPHF[Σ] is updatable complete and updatable zero-knowledge). □

$U_1(\mathtt{w}, \mathtt{lpar} := \mathsf{pk})$		$U_2(\mathtt{w}, \mathtt{lpar} := \mathsf{pk})$
$\mathtt{tc}_{1,\mathsf{up}} := \mathbf{b}_{u_1} \leftarrow\!\!\$\, \mathbb{Z}_p^{n\times k}$		$\mathtt{tc}_{2,\mathsf{up}} := \mathbf{b}_{u_2} \leftarrow\!\!\$\, \mathbb{Z}_p^{n\times k}$
$\mathsf{crs}'_{u_1} \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$		$\mathsf{crs}'_{u_2} \leftarrow \mathsf{Pgen}(1^\lambda, \mathcal{L})$
	$\xrightarrow{\mathsf{crs}'_{u_1}}$	
	$\xleftarrow{\mathsf{crs}'_{u_2}}$	
if $1 \leftarrow \mathsf{crsVer}(\mathsf{pk}, \mathsf{crs}'_{u_2})$:		**if** $1 \leftarrow \mathsf{crsVer}(\mathsf{pk}, \mathsf{crs}'_{u_1})$:
$(\mathsf{crs}'_{1,\mathsf{upd}}, \mathtt{crs}_{1,\mathsf{int}}, \mathtt{tc}_{1,\mathsf{up}}) \leftarrow \mathsf{Upd}(\mathsf{pk}, \mathsf{crs}'_{u_2})$		$(\mathsf{crs}'_{2,\mathsf{upd}}, \mathtt{crs}_{2,\mathsf{int}}, \mathtt{tc}_{2,\mathsf{up}}) \leftarrow \mathsf{Upd}(\mathsf{pk}, \mathsf{crs}'_{u_1})$
$\mathsf{hk}_{u_1} \leftarrow \Sigma.\mathsf{hashkg}(\mathcal{L})$		$\mathsf{hk}_{u_2} \leftarrow \Sigma.\mathsf{hashkg}(\mathcal{L})$
$\mathsf{hp}_{u_1} \leftarrow \Sigma.\mathsf{projkg}(\mathsf{hk}_{u_1}, \mathsf{pk}, \mathsf{crs}'_{1,\mathsf{upd}}, \perp)$		$\mathsf{hp}_{u_2} \leftarrow \Sigma.\mathsf{projkg}(\mathsf{hk}_{u_2}, \mathsf{pk}, \mathsf{crs}'_{2,\mathsf{upd}}, \perp)$
$\mathtt{x}_{u_1} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathtt{w}, r_1)$		$\mathtt{x}_{u_2} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathtt{w}, r_2)$
	$\xrightarrow{(\mathtt{x}_{u_1}, \mathsf{hp}_{u_1}, \mathsf{crs}'_{1,\mathsf{upd}}, \mathtt{crs}_{1,\mathsf{int}})}$	
	$\xleftarrow{(\mathtt{x}_{u_2}, \mathsf{hp}_{u_2}, \mathsf{crs}'_{2,\mathsf{upd}}, \mathtt{crs}_{2,\mathsf{int}})}$	
if $1 \leftarrow \mathsf{crsVer}(\mathsf{pk}, \mathsf{crs}'_{2,\mathsf{upd}})$		**if** $1 \leftarrow \mathsf{crsVer}(\mathsf{pk}, \mathsf{crs}'_{1,\mathsf{upd}})$
$\wedge\, 1 \leftarrow \mathsf{UpdVer}(\mathsf{pk}, \mathsf{crs}'_{u_1}, \mathtt{crs}_{2,\mathsf{int}}, \mathsf{crs}'_{2,\mathsf{upd}})$		$\wedge\, 1 \leftarrow \mathsf{UpdVer}(\mathsf{pk}, \mathsf{crs}'_{u_2}, \mathtt{crs}_{1,\mathsf{int}}, \mathsf{crs}'_{1,\mathsf{upd}})$
$\wedge\, 1 \leftarrow \mathsf{verhp}(\mathsf{hp}_{u_2}, \mathsf{pk}, \mathsf{crs}'_{2,\mathsf{upd}})$:		$\wedge 1 \leftarrow \mathsf{verhp}(\mathsf{hp}_{u_1}, \mathsf{pk}, \mathsf{crs}'_{1,\mathsf{upd}})$:
$\mathsf{pH}_{u_1} \leftarrow \Sigma.\mathsf{projhash}(\mathsf{hp}_{u_1}, \mathsf{pk}, \mathsf{crs}'_{2,\mathsf{upd}}, \mathtt{x}_{u_1}, r_1, \mathtt{w})$		$\mathsf{pH}_{u_2} \leftarrow \Sigma.\mathsf{projhash}(\mathsf{hp}_{u_2}, \mathsf{pk}, \mathsf{crs}'_{1,\mathsf{upd}}, \mathtt{x}_{u_2}, r_2, \mathtt{w})$
$\mathsf{H}_{u_2} \leftarrow \Sigma.\mathsf{hash}(\mathsf{hk}_{u_1}, \mathsf{pk}, \mathtt{x}_{u_2})$		$\mathsf{H}_{u_1} \leftarrow \Sigma.\mathsf{hash}(\mathsf{hk}_{u_2}, \mathsf{pk}, \mathtt{x}_{u_1})$
$\mathsf{sk}_{u_1} \leftarrow \mathsf{pH}_{u_1} \cdot \mathsf{H}_{u_2}$		$\mathsf{sk}_{u_2} \leftarrow \mathsf{pH}_{u_2} \cdot \mathsf{H}_{u_1}$

Fig. 12. Generic Updatable Two-Round PAKE from *uL-TSPHF*