Smooth Zero-Knowledge Hash Functions

Behzad Abdolmaleki[1], Hamidreza Khoshakhlagh[⋆2], and Helger Lipmaa[⋆⋆3,4]

[1] Max Planck Institute for Security and Privacy, Bochum, Germany
[2] Aarhus University, Denmark
[3] Simula UiB, Bergen, Norway
[4] University of Tartu, Estonia

Abstract. We define smooth zero-knowledge hash functions (SZKHFs) as smooth projective hash functions (SPHFs) for which the completeness holds even when the language parameter $\mathtt{lpar}$ and the projection key hp were maliciously generated. We prove that blackbox SZKHF in the plain model is impossible even if $\mathtt{lpar}$ was honestly generated. We then define SZKHF in the registered public key (RPK) model, where both $\mathtt{lpar}$ and hp are possibly maliciously generated but accepted by an RPK server, and show that the CRS-model trapdoor SPHFs of Benhamouda *et al.* are also secure in the weaker RPK model. Then, we define and instantiate subversion-zero knowledge SZKHF in the plain model. In this case, both $\mathtt{lpar}$ and hp are completely untrusted, but one uses non-blackbox techniques in the security proof.

Keywords: plain model, RPK model, SPHF, trapdoor SPHF, subversion zero-knowledge

1 Introduction

Smooth projective hash functions (SPHFs, [CS02]) for an NP language $\mathcal{L}_{\mathtt{lpar}}$ (with corresponding relation $\mathcal{R}_{\mathtt{lpar}}$ such that $\mathcal{L}_{\mathtt{lpar}} = \{\mathtt{x} : \exists \mathtt{w}, (\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}\}$), parametrized by a language parameter $\mathtt{lpar}$, are cryptographic primitives with the following properties. Given $\mathtt{lpar}$ and a word $\mathtt{x}$, one can compute a hash of $\mathtt{x}$ in two different ways: either (i) using a projection key hp (an analogue of a public key), and $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$, as $\mathsf{pH} \leftarrow \mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w})$, or (ii) using a hashing key hk (an analogue of a secret key) and any $\mathtt{x}$, as $\mathsf{H} \leftarrow \mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$. If $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$, then the *completeness* property guarantees that the two ways of computing the hash result in the same value, $\mathsf{pH} = \mathsf{H}$. If $\mathtt{x} \notin \mathcal{L}_{\mathtt{lpar}}$, then the *smoothness* property guarantees that, knowing hp but not hk, one cannot distinguish H from random. SPHFs are useful in many different applications, starting from constructing IND-CCA2 secure cryptosystems [CS02] and password-authenticated key exchange [GL03], and ending with honest-verifier zero knowledge [BBC+13], and non-interactive zero knowledge (NIZK, [ABP15]).

⋆ Funded by the Concordium Foundation under Concordium Blockchain Research Center, Aarhus.
⋆⋆ Partially supported by the Estonian Research Council grant (PRG49).

Several varieties of SPHFs exist. In KV-SPHFs [KV11], hp is created first and then $\mathtt{x}$ can depend on hp. In GL-SPHFs [GL03], the order is opposite. In the current paper, we are primarily interested in the GL-SPHFs. Recent research [BBC+13,ABP15,Ben16] has shown how to construct efficient GL-SPHFs for a large variety of languages. In particular, it is known how to construct GL-SPHFs for the class of algebraic languages $\mathcal{L}_{\boldsymbol{\Gamma},\boldsymbol{\theta}} := \{\mathtt{x} : \exists \mathtt{w}, \boldsymbol{\Gamma}(\mathtt{x}) \cdot \mathtt{w} = \boldsymbol{\theta}(\mathtt{x})\}$, where $\boldsymbol{\Gamma}$ and $\boldsymbol{\theta}$ are $\mathtt{x}$-dependent linear maps, [BBC+13,ABP15,Ben16]. Algebraic languages are quite powerful and include quadratic languages like the languages of the Elgamal encryption of bits, [BBC+13,CH20]. It is also known how to create GL-SPHFs for conjunction and disjunction of two algebraic languages [BBC+13,ABP15,Ben16]. On the other hand, assuming the polynomial hierarchy does not collapse, it is impossible to construct SPHF for NP-complete languages [Ben16].

It is usually assumed that the creator of hp is honest; this explains, e.g., why the SPHF-based two-message zero-knowledge argument of [BBC+13] is honest-verifier only. Benhamouda *et al.* [BBC+13] defined *trapdoor SPHFs* (TSPHFs) as SPHFs where one can verify that the projection key has been generated correctly. Unfortunately, TSPHFs are defined in the strong common reference string (CRS) model, where everybody has to trust the same CRS creator. In many applications, such a universally trusted third party does not exist. This creates another avenue of subversion to which TSPHFs provide no answer: one obtains security against a malicious projection-key creator but not against a malicious CRS creator.

Several recent papers on zero-knowledge arguments [BFS16], including succinct non-interactive arguments of knowledge (SNARKs [ABLZ17,Fuc18]) and quasi-adaptive NIZKs (QA-NIZKs [ALSZ20]), have shown how to achieve either soundness or zero-knowledge even when the public parameters, like the CRS or the public key, have been maliciously subverted. In the case of NIZK, many well-known (im)possibility results exist. E.g., one cannot achieve (say) black-box or even auxiliary-string non-blackbox NIZK in the weak Bare Public Key (BPK, [CGGM00,MR01]; see also Section 2) model [GO94] while efficient no-auxiliary-string non-blackbox zero-knowledge (Sub-ZK) NIZK in the BPK model is possible [ABLZ17,Fuc18,ALSZ20]. Moreover, it is impossible to achieve NIZK that, at the same time, has the properties of subversion-resistant soundness and subversion-resistant zero-knowledge, [BFS16].

We are not aware of similar positive or negative results for SPHFs in the case of trusted or untrusted hp, or of any previous research on the applications of non-blackbox assumptions to SPHF. We emphasize that in the case of SPHFs, this issue is even more critical than in the case of NIZKs: in SPHFs, hp is created by the verifier (who is by definition untrusted by the prover), while in NIZKs, the CRS creator *may* be honest, depending on the application.

Our Contributions. We study SPHFs with untrusted language-parameter and projection-key generator. We say that an SPHF HF is a *smooth zero-knowledge hash function* (SZKHF[5]) if

(i) smoothness holds even for a maliciously generated $\mathtt{lpar}$ (but honestly generated hp), and

(ii) zero-knowledge (i.e., completeness) holds even for a maliciously generated $\mathtt{lpar}$ and a maliciously generated hp, where $\mathtt{lpar}$ and hp are accepted respectively by a public $\mathtt{lpar}$-verification algorithm verpar and a public hp-verification algorithm verhp.

First, we show that SZKHFs are impossible in the plain model. Second, we define SZKHFs in the RPK (registered public key) model, which is weaker than the CRS model. We show that SZKHFs exist in the RPK model. Third, we define Sub-ZK SZKHFs in the plain model, which are SZKHFs without any trust assumption, but similarly to Sub-ZK NIZKs, we use non-blackbox techniques to construct them and show that Sub-ZK GL-SZKHFs exist for all algebraic languages. We focus on the plain model in the main body of the paper, while moving the definitions and constructions for the RPK model to the Appendix B.

On the other hand, Sub-ZK NIZKs are only known in the bare public key model [CGGM00,MR01], which is in stark contrast to our results. In a parallel work, we motivate the difference by showing that Sub-ZK SZKHFs are equivalent to *Sub-ZK deterministic-prover quasi-adaptive two-message zero-knowledge arguments*.

Our Results and Techniques. First, we define blackbox SZKHFs in the plain model without any trust assumptions. Motivated by a classical impossibility result for blackbox two-round zero knowledge in the plain model [GO94], we prove that such SZKHFs are impossible for hard languages even if $\mathtt{lpar}$ is trusted. Thus, one has two options: either (i) allow SZKHFs to rely on non-blackbox assumptions, or (ii) construct SZKHFs in a trust model.

Second, we consider blackbox SZKHFs in the RPK [BCNP04] model (see Appendix B), where each party $\mathcal{P}$ trusts *some* key-registration authority $\mathcal{R}$ and has registered her public key pk with $\mathcal{R}$. If $\mathcal{P}$ is honest, then the secret key sk can be extracted, and pk is correctly distributed. Otherwise, sk can be extracted, but there is no guarantee about its distribution. The RPK model is considerably weaker than the better known CRS model since, in the latter, one assumes that sk is always correctly distributed and that all parties trust the same CRS.

In this case, the zero-knowledge definition is similar to the soundness definition of trapdoor SPHFs (TSPHFs) in [BBC+13], except that the latter is given in the CRS model while we use the weaker RPK model. In addition, motivated by

[5] We considered other terms. This notion corresponds to completeness/projectivity when $\mathtt{lpar}$ and hp are subverted, and thus it could be called *subversion-completeness/subversion-projectivity*. For trapdoor SPHFs, it was called *soundness* in [BBC+13] and, finally, *zero knowledge* in [Ben16]. Zero-knowledge is the most intuitive term since in a typical application of HF; it guarantees that a malicious creator of hp does not learn anything new from seeing pH compared to when she sees H that does not depend on the witness.

recent work on Sub-ZK QA-NIZK [ALSZ20], we assume that $\mathtt{lpar}$ is also created in the RPK model, i.e., its trapdoor is extractable, but there is no guarantee about its distribution; this is in contrast to [BBC+13] that assumed that $\mathtt{lpar}$ is honestly generated. After appropriate tweaking, all known TSPHFs [BBC+13] will become computationally-smooth blackbox SZKHFs in the RPK model. Unfortunately, a TSPHF only shifts the subversion problem: instead of having to trust the generator of hp, one has to trust the generator of $\mathtt{lpar}$ and the RPK; however, in the RPK model, $\mathtt{lpar}$ and the RPK can be handled by different RPK authorities, and there is no need to assume that their trapdoors are correctly distributed. We defer the definitions and constructions of blackbox SZKHF in the RPK model to Appendix B.

Third, inspired by research on Sub-ZK NIZK [BFS16,ABLZ17,Fuc18,ALSZ20], we define Sub-ZK SZKHF in the plain model. Motivated by an impossibility result about two-message zero-knowledge [GO94] and its use in [ALSZ20], we prove that auxiliary-string non-blackbox SZKHF in the plain model is impossible for languages not in BPP, even if $\mathtt{lpar}$ is honestly generated. This impossibility result is strictly stronger than the impossibility result mentioned at the beginning of this subsection. Thus, as in [ALSZ20], Sub-ZK corresponds to *no-auxiliary-string non-blackbox zero-knowledge*. Differently from Sub-ZK NIZK, where one assumes non-blackbox extraction of the secret key, we only require that one can extract $\kappa(\mathsf{hk})$, where κ can be a hard-to-invert bijection. Such a notion of κ-extractability emphasizes the fact that in many applications of SPHFs, it is not essential to extract hk; instead, it suffices to recover a related value $\kappa(\mathsf{hk})$ that can be used to verify efficiently that $\mathsf{projhash}$ was correctly computed.

More formally, let κ be an efficient algorithm, e.g., identity map or exponentiation/bilinear map. An SZKHF is *κ-extractable Sub-ZK* if it supports deterministic algorithms verpar (*language-parameter verification*), verhp (*projection-key verification*) and $\mathsf{simhash}$ (*subversion hash*), s.t. : for each PPT subverter $\mathcal{Z}$ that creates a verpar-accepted $\mathtt{lpar}$ and verhp-accepted hp, there exists a non-blackbox PPT extractor $\mathsf{Ext}_{\mathcal{Z}}$ that outputs $\kappa(\mathsf{hk})$, s.t. $\mathsf{simhash}(\mathtt{lpar}; \kappa(\mathsf{hk}), \mathtt{x}) = \mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w})$ for every $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$. Importantly, compared to blackbox SZKHFs in the RPK model, a Sub-ZK SZKHF in the plain model does not rely on a trusted RPK, and thus, we get full subversion-resistance.

We construct a Sub-ZK SZKHF in the plain model based on SPHFs from DVSs (diverse vector spaces) [BBC+13,Ben16]. Then, we give a construction of computationally smooth blackbox SZKHF in the RPK model based on DVS-based SPHFs. We also present a Sub-ZK SZKHF in the plain model and a blackbox SZKHF in the RPK model, both based on Benhamouda *et al.*'s TSPHFs [BBC+13] in Appendix C.

2 Preliminaries

For a matrix $\boldsymbol{A}$, colspace($\boldsymbol{A}$) is the subspace generated by its columns. Let PPT denote probabilistic polynomial-time. Let vect($\boldsymbol{A}$) be the vectorization of

the matrix $\boldsymbol{A}$. The cokernel of $\boldsymbol{A}$ is defined as $\mathrm{coker}(\boldsymbol{A}) = \{\boldsymbol{a} : \boldsymbol{a}^\top \boldsymbol{A} = \boldsymbol{0}\}$. Let $\lambda \in \mathbb{N}$ be the security parameter. All adversaries will be stateful. For an algorithm $\mathcal{A}$, $\mathrm{im}(\mathcal{A})$ is the image of $\mathcal{A}$ (the set of valid outputs of $\mathcal{A}$), $\mathsf{RND}_\lambda(\mathcal{A})$ is the random tape of $\mathcal{A}$ (for a fixed choice of λ), and $r \leftarrow\!\!\$\, \mathsf{RND}_\lambda(\mathcal{A})$ denotes the random choice of r from $\mathsf{RND}_\lambda(\mathcal{A})$. By $y \leftarrow \mathcal{A}(\mathtt{x}; r)$ we denote that $\mathcal{A}$, given an input $\mathtt{x}$ and a randomizer r, outputs y. 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. Let $\mathsf{negl}(\lambda)$ be an arbitrary negligible function. We write $a \approx_\lambda b$ if $|a - b| \le \mathsf{negl}(\lambda)$. We follow Bellare et al. [BFS16] by using "cryptographic" style in security definitions where all complexity (adversaries, algorithms, assumptions) is uniform, but the security (say, soundness) is quantified over all inputs chosen by the adversary.

A bilinear group generator $\mathsf{Pgen}(1^\lambda)$ returns $\mathsf{p} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, [1]_1, [1]_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are three additive cyclic groups of prime order p, $[1]_\iota$ is a generator of $\mathbb{G}_\iota$ for $\iota \in \{1, 2, T\}$ with $[1]_T = \hat{e}([1]_1, [1]_2)$, and $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate efficiently computable bilinear pairing. We assume λ is implicitly described by p, and as in [BFS16], we assume that p is a deterministic function of λ and thus cannot be subverted. (This is usually the case in practice.) We require the bilinear pairing to be Type-3, that is, we assume that there is no efficient isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We use the additive implicit notation of [EHK+13], that is, we write $[a]_\iota$ to denote $a[1]_\iota$ for $\iota \in \{1, 2, T\}$. We denote $\hat{e}([a]_1, [b]_2)$ by $[a]_1 \bullet [b]_2$. Thus, $[a]_1 \bullet [b]_2 = [ab]_T$. We freely use the bracket notation together with matrix notation; for example, if $\boldsymbol{AB} = \boldsymbol{C}$ then $[\boldsymbol{A}]_1 \bullet [\boldsymbol{B}]_2 = [\boldsymbol{C}]_T$. We also assume that $[\boldsymbol{A}]_2 \bullet [\boldsymbol{B}]_1 := ([\boldsymbol{B}]_1^\top \bullet [\boldsymbol{A}]_2^\top)^\top = [\boldsymbol{AB}]_T$.

Algebraic Languages. Let p be system parameters, including say the description of a bilinear group. Let $\mathtt{lpar} = (\boldsymbol{\Gamma}, \boldsymbol{\theta}, \boldsymbol{\lambda})$, where $\boldsymbol{\Gamma}, \boldsymbol{\theta}, \boldsymbol{\lambda}$ are all linear maps in their their inputs. More precisely, $\boldsymbol{\Gamma}(\mathtt{x})$ is an $n \times k$ matrix, $\boldsymbol{\theta}(\mathtt{x})$ is an n-dimensional vector, and $\boldsymbol{\lambda}(\mathtt{x}, \mathtt{w})$ is a k-dimensional vector. Moreover, different coefficients of $\boldsymbol{\theta}(\mathtt{x})$, $\boldsymbol{\Gamma}(\mathtt{x})$, and $\boldsymbol{\lambda}(\mathtt{x}, \mathtt{w})$ can belong to different algebraic structures (most commonly, given a bilinear group $\mathsf{p} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e})$, either to $\mathbb{Z}_p$, $\mathbb{G}_1$, $\mathbb{G}_2$, or $\mathbb{G}_T$) as long as the equation $\boldsymbol{\theta}(\mathtt{x}) = \boldsymbol{\Gamma}(\mathtt{x}) \cdot \boldsymbol{\lambda}(\mathtt{x}, \mathtt{w})$ is "well-typed". E.g., the equation $\begin{pmatrix} [\theta_1]_T \\ [\theta_2]_1 \end{pmatrix} = \begin{pmatrix} [\Gamma_{11}]_T & [\Gamma_{12}]_2 \\ [\Gamma_{21}]_1 & \Gamma_{22} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ [\lambda_2]_1 \end{pmatrix}$ is well-typed. We omit the subscript $\mathtt{lpar}$ if it is clear from the context. Define

$$\mathcal{L}_{\mathtt{lpar}} = \{\mathtt{x} : \exists \mathtt{w}, \boldsymbol{\theta}(\mathtt{x}) = \boldsymbol{\Gamma}(\mathtt{x}) \cdot \boldsymbol{\lambda}(\mathtt{x}, \mathtt{w})\} \ . \tag{1}$$

Let $\mathcal{R}_{\mathcal{L}} = \{(\mathtt{x}, \mathtt{w}) : \boldsymbol{\theta}(\mathtt{x}) = \boldsymbol{\Gamma}(\mathtt{x}) \cdot \boldsymbol{\lambda}(\mathtt{x}, \mathtt{w})\}$ be the corresponding witness-relation. Languages of the form Eq. (1) have been studied at least since [BBC+13], and they called *algebraic*[6] in [CH20]. All linear languages are algebraic, but algebraic languages also include non-linear languages. E.g., the language of Elgamal encryptions of bits is algebraic [BBC+13]; in this case, $\boldsymbol{\Gamma}(\mathtt{x})$ depends on $\mathtt{x}$.

Projective Hash Functions. Let $\mathcal{L}_{\mathtt{lpar}} \subset \mathcal{X}_{\mathtt{lpar}}$ be a language parametrized by $\mathtt{lpar}$ (the language parameter), where $\mathcal{X}_{\mathtt{lpar}}$ is the underlying domain, e.g., a group. Let $\mathcal{R}_{\mathtt{lpar}}$ be the witness-relation defined by $\mathcal{L}_{\mathtt{lpar}} = \{\mathtt{x} : \exists \mathtt{w}, (\mathtt{x}, \mathtt{w}) \in$

[6] Couteau and Hartmann [CH20] considered $\boldsymbol{\lambda}(\mathtt{x}, \mathtt{w}) := \mathtt{w}$ only; however, one can just redefine the witness to contain all elements of $\boldsymbol{\lambda}(\mathtt{x}, \mathtt{w})$

$\mathcal{R}_{\mathtt{lpar}}\}$. A projective hash function (PHF, [CS02]) for $\{\mathcal{L}_{\mathtt{lpar}}\}$ is a tuple of PPT algorithms $\mathsf{HF} = (\mathsf{Pgen}, \mathsf{setup.lpar}, \mathsf{hashkg}, \mathsf{projkg}, \mathsf{hash}, \mathsf{projhash})$, where

$\mathsf{Pgen}(1^\lambda)$**:** Takes a security parameter λ and generates the global parameters p.

$\mathsf{setup.lpar}(\mathsf{p})$**:** sets up the language parameters $\mathtt{lpar}$. $\mathtt{lpar}$ contains p and some public parameters specifying the relation (e.g., an encryption key).

$\mathsf{hashkg}(\mathtt{lpar})$**:** Inputs a language parameter $\mathtt{lpar}$. It generates and outputs a hashing key hk for $\mathcal{L}_{\mathtt{lpar}}$.

$\mathsf{projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$**:** Inputs a language parameter $\mathtt{lpar}$, a hashing key hk, and possibly a word $\mathtt{x} \in \mathcal{X}_{\mathtt{lpar}}$. It outputs deterministically a projection key hp.

$\mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$**:** Inputs a language parameter $\mathtt{lpar}$, a hashing key hk, and a word $\mathtt{x} \in \mathcal{X}_{\mathtt{lpar}}$. It outputs deterministically a hash value H.

$\mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w})$**:** inputs a language parameter $\mathtt{lpar}$, a projection key hp, and $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$. It outputs deterministically a projected hash value pH.

The set of hash values is called the *range* of HF and is denoted by $\mathsf{HashSet}$. We assume $\mathsf{HashSet}$ is an efficiently sampleable set that has size, exponential in λ. To shorten notation, we will denote the sequence "$\mathsf{hk} \leftarrow \mathsf{hashkg}(\mathtt{lpar}); \mathsf{hp} \leftarrow \mathsf{projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$" by $(\mathsf{hp}, \mathsf{hk}) \leftarrow \mathsf{kgen}(\mathtt{lpar}; \mathtt{x})$.

A distribution $\mathcal{D}_{\mathsf{p}}$ (e.g., the output distribution of $\mathsf{setup.lpar}(\mathsf{p})$) on $\mathcal{L}_{\mathtt{lpar}}$ is *witness-sampleable* [JR13] if there exists a PPT algorithm $\mathsf{setup.ltrap}(\mathsf{p})$ that samples $(\mathtt{lpar}, \mathtt{ltrap}) \in \mathcal{R}_{\mathsf{p}}$ such that $\mathtt{lpar}$ is distributed according to $\mathcal{D}_{\mathsf{p}}$, and membership of $\mathtt{x}$ in *the parameter language* $\mathcal{L}_{\mathtt{lpar}}$ can be verified in PPT given $\mathtt{ltrap}$. We always assume that $\mathtt{lpar}$ can be efficiently computed from $\mathtt{ltrap}$. In SPHF-related research, $\mathcal{D}_{\mathsf{p}}$ is often assumed to be witness-sampleable, even if it is not always necessary. Couteau and Hartmann [CH20] extended the definition of witness-sampleable languages to all algebraic languages.

HF is *perfectly complete* if for all $\mathtt{lpar} \in \mathrm{im}(\mathsf{setup.lpar}(\mathsf{p}))$, $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$, and $(\mathsf{hp}, \mathsf{hk}) \in \mathrm{im}(\mathsf{kgen}(\mathtt{lpar}; \mathtt{x}))$, $\mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x}) = \mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w})$.

There are at least three types of smooth PHFs (SPHFs). Intuitively, in GL-SPHF [GL03], security is required even when hp maliciously depends on $\mathtt{x}$. On the other hand, in KV-SPHF [KV11], security is required even when $\mathtt{x}$ can maliciously depend on hp. The third type is CS-SPHF, [CS02]; we will not discuss CS-SPHFs in what follows. See [Ben16, Section 2.5] for more information.

A PHF HF for a language $\mathcal{L} \subseteq \mathcal{X}$ is *ε-GL-smooth (an ε-GL-SPHF)* if for any $\mathtt{lpar}$ and any word $\mathtt{x} \in \mathcal{X}_{\mathtt{lpar}} \setminus \mathcal{L}_{\mathtt{lpar}}$, the following distributions are ε-close:

$$\{(\mathsf{hp}, \mathsf{H}) : (\mathsf{hp}, \mathsf{hk}) \leftarrow \mathsf{kgen}(\mathtt{lpar}; \mathtt{x}); \mathsf{H} \leftarrow \mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})\}$$
$$\{(\mathsf{hp}, \mathsf{H}) : (\mathsf{hp}, \mathsf{hk}) \leftarrow \mathsf{kgen}(\mathtt{lpar}; \mathtt{x}); \mathsf{H} \leftarrow_{\$} \mathsf{HashSet}\}$$

A PHF is *GL-smooth* if it is ε-GL-smooth with ε negligible in λ.

HF for $\mathcal{L} \subseteq \mathcal{X}$ is *ε-KV-smooth (an ε-KV-SPHF)* if for any $\mathtt{lpar}$ and any (not necessarily computable in polynomial-time) function f from the set of possible projection keys hp to $\mathcal{X}_{\mathtt{lpar}} \setminus \mathcal{L}_{\mathtt{lpar}}$, the following distributions are ε-close:

$$\{(\mathsf{hp}, \mathsf{H}) : (\mathsf{hp}, \mathsf{hk}) \leftarrow \mathsf{kgen}(\mathtt{lpar}); \mathsf{H} \leftarrow \mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, f(\mathsf{hp}))\}$$
$$\{(\mathsf{hp}, \mathsf{H}) : (\mathsf{hp}, \mathsf{hk}) \leftarrow \mathsf{kgen}(\mathtt{lpar}); \mathsf{H} \leftarrow_{\$} \mathsf{HashSet}\}$$

A PHF is *KV-smooth* if it is ε-KV-smooth with ε negligible in λ. Since projkg does not depend on $\mathtt{x}$ in this case, we often omit $\mathtt{x}$ as an argument for projkg.

For all (subset-membership-hard) algebraic languages, one can construct an efficient SPHF, [BBC+13,ABP15,Ben16], s.t. the hash value belongs to a source group, $\mathbb{G}_1$ or $\mathbb{G}_2$.

3 Defining SZKHF

While completeness of SPHF, defined for honestly generated hp, is sufficient in many applications, it is natural to ask what happens if hp was generated maliciously. Consider, e.g., an application of SPHFs in the construction of zero-knowledge proof systems. One can use SPHFs to design two-message honest-verifier zero-knowledge proof systems [BBC+13] and non-interactive zero-knowledge (NIZKs) argument systems in the CRS model [ABP15]. In the former case, the need to trust the hp generator translates to the need to trust the verifier who creates hp (hence, one gets honest-verifier zero-knowledge). While [BBC+13] showed how to obtain two-message zero-knowledge proof systems, they did it by introducing a trusted CRS generator. In this case and the case of SPHF-based NIZKs, [ABP15], the need to trust the hp generator is transformed to the need to trust the CRS generator.

The CRS model assumes the existence of a universally trusted CRS creator who creates the CRS from the correct distribution and does not leak any information. Unfortunately, NIZK in the plain model, and even auxiliary-string NIZK in the BPK [CGGM00,MR01] model, is impossible, [GO94]. One can construct efficient no-auxiliary-string non-blackbox zero-knowledge NIZK in the BPK model based on SNARKs and QA-NIZKs [ABLZ17,Fuc18,ALSZ20] assuming there exists a public BPK verification procedure PKV and, in the case of QA-NIZK [ALSZ20], a public language parameter verification procedure verpar. No-auxiliary-string non-blackbox implies that, given the BPK pk is accepted by PKV, one can use an adversary-dependent extractor to extract the trapdoor of pk, and, in the case of QA-NIZK, $\mathtt{lpar}$ is accepted by verpar. For the extraction to succeed, it is required that the adversary has no auxiliary string since an auxiliary string could encode a pk for which she does not know the trapdoor.

Since SPHFs can be used to construct NIZKs [ABP15], one can hope that some of the known (im)possibility results about NIZKs can be translated to the case of SPHFs. However, this is not evident, in particular since there is no prior work on non-blackbox SPHFs or SPHFs in different trust models, except [BBC+13] that only considers SPHFs in the CRS model. Thus, we need to use known (im)possibility results about *two-message* zero-knowledge argument systems.

We approach the question of untrusted $\mathtt{lpar}$ and hp systematically. We will define a stronger version of completeness (*zero-knowledge*) of an SPHF that guarantees that even if $\mathtt{lpar}$ and hp are created maliciously then either

(i) one detects that this is the case, or
(ii) if $(\mathtt{x},\mathtt{w}) \in \mathcal{R}_{\mathcal{L}}$ then $\mathsf{hash}(\mathtt{lpar};\mathsf{hk},\mathtt{x}) = \mathsf{projhash}(\mathtt{lpar};\mathsf{hp},\mathtt{x},\mathtt{w})$.

Additionally, we define a stronger version of smoothness, called *Sub-PAR smoothness* of an SPHF which guarantees that the smoothness holds even if $\mathtt{lpar}$ (but not hp) is created maliciously. A *smooth zero-knowledge hash function* (SZKHF) is an SPHF that satisfies zero-knowledge and Sub-PAR smoothness. The precise model (Sub-PAR smoothness and Sub-ZK) is motivated by the model used in [ALSZ20] in the case of QA-NIZK. However, since SZKHFs are related not to QA-NIZK but to a flavour of two-message zero-knowledge argument systems, and thus the completeness of this model has to established separately in the case of SZKHFs.

We will consider SZKHFs in the following three models:

Blackbox zero-knowledge (ZK) in the plain model: ZK holds without the use of non-blackbox techniques or trust assumptions. We show that blackbox SZKHF in the plain model is impossible for languages not in BPP, even if $\mathtt{lpar}$ was honestly generated and auxiliary input is not allowed.[7]

Blackbox ZK in the RPK model: ZK holds without the use of non-blackbox techniques but one relies on the RPK model. In this case, SZKHF is a variant of the definition of TSPHFs from [BBC+13] that however were defined in the stronger CRS model. More precisely, both $\mathtt{lpar}$ and hp can be untrusted but they need to be accepted by an RPK server. (Thus, one can extract $\mathtt{ltrap}$ and $\mathtt{td}$ in the security proof.) On the other hand, [BBC+13] assumes that $\mathtt{lpar}$ and the CRS are correctly distributed. Known TSPHFs can be tweaked to be SZKHF in this sense but one still has the issue of the subversion of both $\mathtt{lpar}$ and the $\mathtt{rpk}$.

Non-blackbox ZK in the plain model: ZK is proven by non-blackbox techniques in the plain model. Here, the SZKHF definiton is related to that of the subversion zero-knowledge (Sub-ZK) Sub-PAR smooth QA-NIZKs [ALSZ20]. We show that auxiliary-string non-blackbox SZKHF in the plain model is impossible for languages not in BPP, even if $\mathtt{lpar}$ was honestly generated.

In all three cases, we will assume that there exist deterministic PPT algorithms verpar and verhp, such that correctness holds even if $\mathtt{lpar}$ and hp were maliciously constructed as long as verpar accepts $\mathtt{lpar}$ and verhp accepts $(\mathtt{lpar}; \mathsf{hp}, \mathtt{x})$. (Note that verhp takes the input $\mathtt{x}$ only when we have a GL-SPHF.) We assume that verpar (resp., verhp) accepts all correctly generated language parameters (resp., projection keys). The existence of verhp for SPHFs was first postulated by Benhamouda *et al.* [BBC+13] who used it to obtain trapdoor SPHFs. An analogous algorithm $\mathtt{CV}$ for NIZKs was (independently) postulated for NIZKs in [ABLZ17] and played a key part in their definition of Sub-ZK NIZK in the CRS model. We are not aware of any previous definition of verpar in the case of SPHFs; in the case of QA-NIZKs, it was first done in [ALSZ20].

[7] In the case of blackbox ZK in the plain model, we will give the definition only for honestly generated $\mathtt{lpar}$: since we will show that this definition is impossible to achieve, this will make our result only stronger.

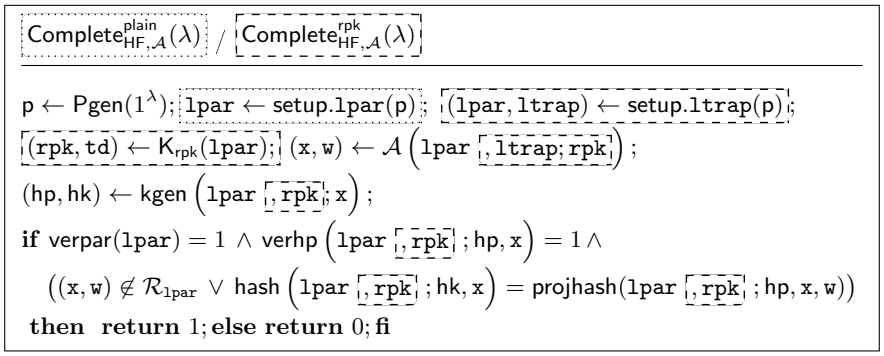

Fig. 1. Completeness experiments in Definitions 1, 2 and 4. The dashed-box / dotted-box part is only present in the dashed-boxed / dotted-boxed experiment.

In the rest of the paper, we only consider GL-SZKHFs: security definitions in the case of GL-SZKHFs and KV-SZKHFs differ in small technical details that mostly just make it more difficult to parse the definitions.

3.1 Blackbox SZKHF in the Plain Model

Next, we define blackbox GL-SZKHF in the plain model. We prove that, even if `lpar` was honestly generated, this definition can only be satisfied for languages in BPP. In Definitions 1, 2 and 4, we postulate the existence of a deterministic algorithm $\mathsf{simhash}$, such that for any $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$, $\mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w}) = \mathsf{simhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x})$. Here, $\mathsf{simhash}$ does not get either the RPK trapdoor `td` or hk (or even $\kappa(\mathsf{hk})$, where κ is a possibly hard-to-invert bijection) as an input.

As in the case of TSPHFs [BBC+13], we assume only computational smoothness. Moreover, in the definition of smoothness, we only consider honestly generated `lpar`, and consider security in the case when $\mathcal{A}$ does not have access to `ltrap` (and thus the definition is not restricted to witness-sampleable languages). All these changes only make our impossibility result stronger.

A *GL-SZKHF in the plain model* is a PHF together with new deterministic algorithms verpar, verhp and $\mathsf{simhash}$ defined as follows:

- $\mathsf{verhp}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x})$ outputs 1 if hp is a valid projection key and 0 otherwise.
- $\mathsf{verpar}(\mathtt{lpar})$: outputs 1 if `lpar` is well-formed and 0 otherwise.
- $\mathsf{simhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x})$ returns the trapdoor hash value of $\mathtt{x}$, given hp.

Definition 1. *A GL-SZKHF* $\mathsf{HF} = (\mathsf{Pgen}, \mathsf{setup.ltrap}, \mathsf{hashkg}, \mathsf{projkg}, \mathsf{hash}, \mathsf{projhash}, \mathsf{verhp}, \mathsf{verpar}, \mathsf{simhash})$ *in the plain model satisfies the following properties, for the experiments depicted in Figs. 1 to 3.*

Perfect completeness: *for all* λ, *PPT* $\mathcal{A}$, $\Pr[\mathsf{Complete}^{\mathsf{plain}}_{\mathsf{HF},\mathcal{A}}(\lambda) = 1] = 1$.

Computational (blackbox) smoothness: $\forall$ *PPT* $\mathcal{A}$, $\Pr[\mathsf{Smooth}^{\mathsf{bb\text{-}plain}}_{\mathsf{HF},\mathcal{A}}(\lambda) = 1] \approx_\lambda \frac{1}{2}$. *SZKHF is* statistically smooth *if this holds for all unbounded* $\mathcal{A}$.

Composable (blackbox) ZK: $\forall$ *PPT subverters* $\mathcal{Z}$, *unbounded* $\mathcal{A}$, $\Pr[\mathsf{ZK}^{\mathsf{bb\text{-}plain}}_{\mathsf{HF},\mathcal{Z},\mathcal{A}}(\lambda) = 1] \approx_\lambda \frac{1}{2}$.

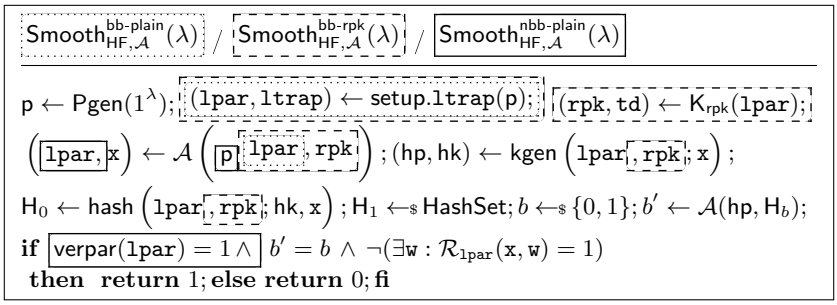

Fig. 2. Smoothness experiments in Definitions 1, 2 and 4. The boxed / dashed-box / dotted-box part is only present in the boxed / dashed-boxed / dotted-boxed experiment.

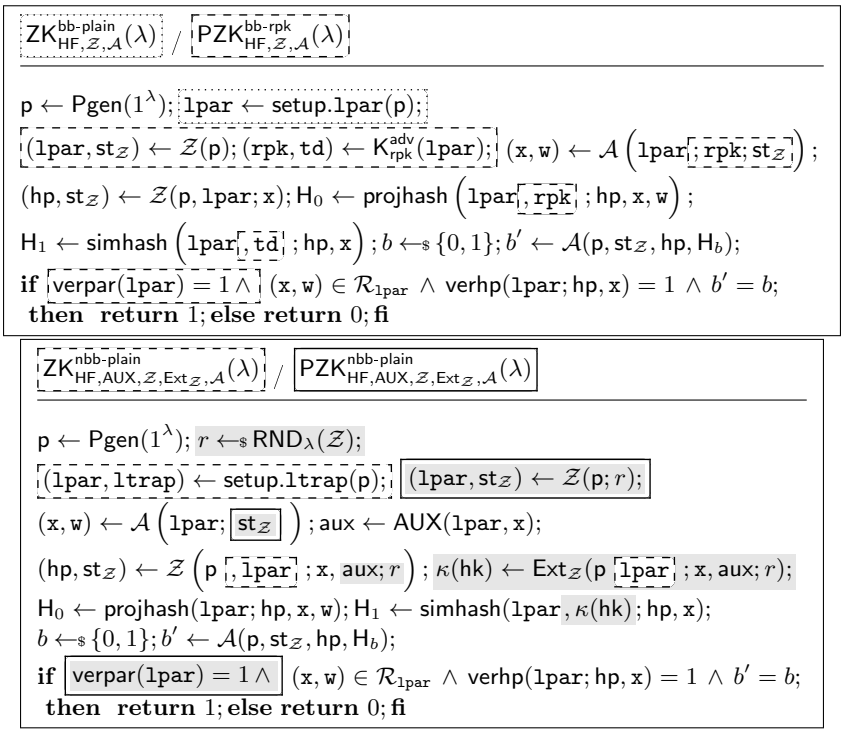

Fig. 3. (Persistent) zero-knowledge experiments in Definitions 1, 2 and 4. The boxed / dashed-box / dotted-box part is only present in boxed / dashed-boxed / dotted-boxed experiments. Also, gray background marks differences compared to $\mathsf{ZK}^{\mathsf{bb\text{-}plain}}_{\mathsf{HF},\mathcal{Z},\mathcal{A}}(\lambda)$.

(Recall that p is a deterministic function of λ.) Note that unbounded $\mathcal{A}$ creates $(\mathtt{x}, \mathtt{w})$ and only $\mathtt{x}$ is passed to bounded subverter $\mathcal{Z}$; this is necessary since in the case of GL-SZKHF, hp can depend on $\mathtt{x}$. We do not allow $\mathcal{A}$ to transmit any

other information. We consider $\mathcal{A}$ only to be successful if $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$. Sadly, it is easy to show that definition 1 can only be satisfied for $\mathcal{L}_{\mathtt{lpar}} \in \mathsf{BPP}$.

Lemma 1. *Let* HF *be a computationally smooth and composable ZK GL-SZKHF in the plain model for* $\mathcal{L}_{\mathtt{lpar}}$ *under blackbox assumptions. Then* $\mathcal{L}_{\mathtt{lpar}} \in \mathsf{BPP}$.

This lemma is a corollary of Theorem 1 from Section 3.2, but for the sake of completeness, we will next give a direct proof. A simple modification of the proof also shows the impossibility of KV-SZKHFs in the plain model.

Proof. Let HF be a computationally-smooth and composable ZK GL-SZKHF in the plain model for $\mathcal{L}_{\mathtt{lpar}}$. We describe $\mathcal{B}$, the BPP adversary for deciding $\mathcal{L}_{\mathtt{lpar}}$ as follows:

$\mathcal{B}(\mathtt{lpar}, \mathtt{x})$

$(\mathsf{hp}, \mathsf{hk}) \leftarrow \mathsf{kgen}(\mathtt{lpar}; \mathtt{x}); b_{\mathcal{A}} \leftarrow\!\!\$\ \{0,1\};$
$\mathsf{H}_0 \leftarrow \mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x}); \mathsf{H}_1 \leftarrow \mathsf{simhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x});$
if $\mathsf{H}_0 = \mathsf{H}_1$ **then** $b' \leftarrow 0$; **else** $b' \leftarrow 1$; **fi**
return b';

The challenger $\mathcal{C}$ of the BPP-decision game samples $\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda)$, $\mathtt{lpar} \leftarrow \mathsf{setup.lpar}(\mathsf{p})$, $b \leftarrow\!\!\$\ \{0,1\}$, $\mathtt{x}_0 \leftarrow\!\!\$\ \mathcal{L}_{\mathtt{lpar}}$, $\mathtt{x}_1 \leftarrow\!\!\$\ \mathcal{X} \setminus \mathcal{L}_{\mathtt{lpar}}$. For $\mathtt{x} \leftarrow \mathtt{x}_b$, $\mathcal{C}$ sends $(\mathtt{lpar}; \mathtt{x})$ to $\mathcal{B}$ who returns b'.

The soundness of $\mathcal{B}$ follows directly from the computational-smoothness of HF. For any $\mathtt{x}_b \notin \mathcal{L}_{\mathtt{lpar}}$, $\mathcal{B}$ will output $b' = 1$ with probability at least $1 - \varepsilon_{sm}$. Also, the Sub-ZK property of the HF guarantees the completeness of $\mathcal{B}$. Thus:

$$\begin{aligned}\Pr[b' = b] &= \left(\Pr[b' = 0 | b = 0] + \Pr[b' = 1 | b = 1]\right)/2 \\ &= \Pr[\mathsf{H}_0 = \mathsf{H}_1 | \mathtt{x} = \mathtt{x}_0]/2 + \Pr[\mathsf{H}_0 \neq \mathsf{H}_1 | \mathtt{x} = \mathtt{x}_1]/2 \\ &\geq \tfrac{1}{2} + \tfrac{1-\varepsilon_{sm}}{2} = 1 - \tfrac{\varepsilon_{sm}}{2} \quad .\end{aligned}$$

Thus, $\mathcal{B}$ has non-negligible advantage in deciding $\mathcal{L}_{\mathtt{lpar}}$. □

3.2 Sub-ZK SZKHF in the Plain Model

In Appendix B, we define SZKHF in the RPK model and give a construction of computationally smooth blackbox SZKHF in this model. Now, we consider the second direction of weakening Definition 1, namely, that of using non-blackbox techniques. To this end, we modify the Sub-ZK definition of QA-NIZKs by Abdolmaleki *et al.* [ALSZ20] to the case of SPHFs. Differently from Appendix B, to facilitate reading by readers who come from the SPHF background, we will first motivate the security definition.

Briefly, [ALSZ20] defines QA-NIZKs in the Bare Public Key (BPK) model, assuming that the public key pk and possibly lpar are created by a malicious subverter $\mathcal{Z}$. They define Sub-PAR soundness (soundness even if both lpar and

pk are maliciously created), Sub-ZK (ZK, even if pk is maliciously created), and persistent Sub-ZK (ZK, even if both lpar and pk are maliciously created).

According to [BFS16], independently of how lpar was generated, one cannot get at the same time Sub-SND (subversion-soundness, soundness if pk is maliciously generated) and Sub-ZK. [ALSZ20] constructed a Sub-PAR sound and persistent Sub-ZK QA-NIZK. Moreover, [ALSZ20] noted that Sub-ZK (QA-)NIZK in the CRS model is the same as *no-auxiliary-string non-blackbox* (QA-)NIZK in the weak BPK model. The Sub-ZK definition of [ALSZ20] is motivated by the fact that auxiliary-string non-blackbox [GO94] NIZK in the BPK model is impossible.

More precisely, a Sub-ZK QA-NIZK in the BPK model [ALSZ20] guarantees that if a malicious subverter $\mathcal{Z}$ creates lpar and pk that are accepted by a verpar (lpar-verification) and PKV (public-key verification), respectively, then there exists a non-blackbox extractor $\mathsf{Ext}_{\mathcal{Z}}$ that extracts the secret key sk that corresponds to pk. After that, sk can be used to run the original CRS-model simulator Sim that works in the case pk is generated honestly. Hence, one obtains non-blackbox ZK.

Next, we consider Sub-ZK SZKHFs *in the plain model* that are motivated by QA-NIZKs in the BPK model. In the case of SZKHF, we have a hp instead of the pk, hk instead of sk, verhp instead of PKV, projhash instead of the prover, and simhash instead of the simulator. Intuitively, since in many applications, hp is generated by the SZKHF verifier (the party who checks that hash and projhash results in the same values), a Sub-ZK SZKHF works in the plain model, i.e., without any trust assumptions at all. This is a fundamental difference compared to Sub-ZK SNARKs and QA-NIZKs where one has to rely on some trust assumption due to the use of the BPK.

As in [ALSZ20], we define an efficient lpar-verification algorithm verpar (denoted by PARV in [ALSZ20]) which checks whether lpar is well-formed. Following the definition of SZKHFs in the RPK model (Appendix B), we allow one to extract a function $\kappa(\mathsf{hk})$ of hk instead of hk itself. In general, κ may be the identity or a one-way function, e.g., $\kappa(\mathsf{hk}) = [\mathsf{hk}]_2$. In the latter case, it may not be possible to efficiently recover hk from $\kappa(\mathsf{hk})$. Due to this, we require that $\mathsf{simhash}(\mathtt{lpar}; \kappa(\mathsf{hk}), \mathsf{hp}, \mathtt{x}) = \mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$ for all lpar, hk, and x.

By analogy to [ALSZ20], we obtain Definition 2. It is a variant of the definition of Sub-ZK QA-NIZKs, with syntactic differences caused by differences between SPHFs and NIZKs. On top of it, the definition is for GL-SZKHFs, which means that in the definition of ZK, a subverted hp can depend on input x chosen by the adversary before hp itself is chosen. In comparison, QA-NIZKs are related to KV-SZKHFs where x depends on hp. Sub-ZK NIZK is impossible in the plain model, [BFS16]. On the other hand, as we will show in Section 4, Sub-ZK SZKHFs are possible in the plain model. We define separately auxiliary-string and no-auxiliary-string non-blackbox ZK in the plain model; this is motivated by Theorem 1 that states that the former is impossible for languages not in BPP. In the case of auxiliary-string non-blackbox ZK, we allow the auxiliary input to

be generated by a PPT algorithm AUX called *auxiliary string machine* which takes the language parameter $\mathtt{lpar}$ as input and returns aux.

Definition 2. *A (no-)auxiliary-string non-blackbox zero knowledge GL-SZKHF* $\mathsf{HF} = (\mathsf{Pgen}, \mathsf{setup}.\mathtt{ltrap}, \mathsf{hashkg}, \mathsf{projkg}, \mathsf{hash}, \mathsf{projhash})$ *in the plain model satisfies the following properties for deterministic polynomial-time algorithms* verpar, verhp, $\mathsf{simhash}$, κ, *and the experiments depicted in Figs. 1 to 3.*

Perfect completeness: *for any* λ, *PPT* $\mathcal{A}$, $\Pr[\mathsf{Complete}^{\mathsf{plain}}_{\mathsf{HF},\mathcal{A}}(\lambda) = 1] = 1$.

Computational Sub-PAR (non-blackbox) smoothness: *for any PPT* $\mathcal{A}$, $\Pr[\mathsf{Smooth}^{\mathsf{nbb\text{-}plain}}_{\mathsf{HF},\mathcal{A}}(\lambda) = 1] \approx_\lambda \frac{1}{2}$. *SZKHF is* statistically Sub-PAR smooth *if the same holds for any unbounded* $\mathcal{A}$.

Composable κ-extractable (no-)auxiliary-string non-blackbox ZK: *For any PPT subverter* $\mathcal{Z}$, *there exists a PPT extractor* $\mathsf{Ext}_{\mathcal{Z}}$, *s.t. for any PPT auxiliary string machine* AUX *and unbounded* $\mathcal{A}$, $\Pr[\mathsf{ZK}^{\mathsf{nbb\text{-}plain}}_{\mathsf{HF},\mathsf{AUX},\mathcal{Z},\mathsf{Ext}_{\mathcal{Z}},\mathcal{A}}(\lambda) = 1] \approx_\lambda \frac{1}{2}$. *In the no-auxiliary-string case,* AUX *always outputs* ϵ *(the empty string).*

Composable κ-extractable (no-)auxiliary-string non-blackbox persistent ZK: *For any PPT subverter* $\mathcal{Z}$, *there exists a PPT extractor* $\mathsf{Ext}_{\mathcal{Z}}$, *s.t. for any PPT auxiliary string machine* AUX *and unbounded* $\mathcal{A}$, $\Pr[\mathsf{PZK}^{\mathsf{nbb\text{-}plain}}_{\mathsf{HF},\mathsf{AUX},\mathcal{Z},\mathsf{Ext}_{\mathcal{Z}},\mathcal{A}}(\lambda) = 1] \approx_\lambda \frac{1}{2}$. *In the no-auxiliary-string case,* AUX *always outputs* ϵ.

HF is *extractable* if κ is the identity function; then, for $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$, $\mathsf{simhash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x}) = \mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w})$. Differently from F-extractability [BCKL08] that limits applications compared to just extractability, we use κ-extractability only in the Sub-ZK proof and thus it has no negative effect.

As explained in Appendix B, Abdolmaleki *et al.* [ALSZ20] (see also Appendix B) defined separately ZK and persistent ZK for QA-NIZK, and showed that ZK does not follow from persistent ZK since in the latter one can use a knowledge assumption to extract $\mathtt{ltrap}$ that is not available in the former. The same problem holds in the case of SZKHFs, and thus in the security proofs, one has to prove separately that a Sub-ZK SZKHF satisfies both ZK and persistent ZK.

Motivated by applications in SNARKs, Abdolmaleki *et al.* [ALSZ20] defined the notion of knowledge-soundness in the case $\mathcal{L}_{\mathtt{lpar}} = \mathcal{X}$ is the trivial language. One can similarly define knowledge-smoothness when $\mathcal{L}_{\mathtt{lpar}} = \mathcal{X}$; we decided not to do it since we already have too many new definitions.

Impossibility of auxiliary-string SZKHF in the plain model. Goldreich and Oren [GO94, Thm. 4.4] proved that two-round non-uniform auxiliary-string computational zero-knowledge proof (and also argument) systems do not exist for languages outside BPP. We modify Thm. 4.4 of [GO94] to prove a similar result about GL-SZKHFs. Note that this result is strictly stronger than Lemma 1.

Motivated by this connection, we show in a parallel work that no-auxiliary-string non-blackbox GL-SZKHFs and quasi-adaptive two-message zero-knowledge (QA-2MZK) arguments are in one-to-one correspondence. A similar

result holds in the case of blackbox and no-auxiliary-string non-blackbox GL-SZKHFs and QA-2MZK arguments. We omit further discussion.

Theorem 1. *Let κ be a one-to-one map,* HF *be an auxiliary-string non-blackbox ZK GL-SZKHF in the plain model for $\mathcal{L}_{\mathtt{lpar}}$. Then $\mathcal{L}_{\mathtt{lpar}} \in \mathsf{BPP}$ for all* $\mathtt{lpar}$.

We defer the proof to Appendix E.1.

4 Constructing SZKHF

In Section 3.1, we proved that blackbox SZKHFs in the plain model are restricted to languages in BPP. Thus, one must either use a preprocessing model (as defined in [BBC+13]; see Appendix C) or rely on some non-blackbox technique (as defined in Section 3.2). As we already mentioned, the use of the CRS model as in [BBC+13] (or the weaker RPK model, as in Appendix B) is not completely satisfactory since one essentially shifts the problem of protecting against a subverted hp-generator to the problem of protecting against a subverted $\mathtt{crs}/\mathsf{pk}$-generator. As always in cryptography, the end goal is not to have any trust at all whenever possible.

We first recall the notion of DVS-based SPHF. Then, we construct a Sub-ZK SZKHF in the plain model based on DVS-based SPHF. Lastly, we give a constructions of computationally smooth blackbox SZKHF in the RPK model based on DVS-based SPHFs in Appendix B. We also present a Sub-ZK SZKHF in the plain model and a blackbox SZKHF in the RPK model, both based on Benhamouda *et al.*'s TSPHFs [BBC+13] in Appendix C.

4.1 Preliminaries: DVS-Based SPHFs

Benhamouda *et al.* [BBC+13,Ben16] defined diverse vector spaces (DVSs). We will not formally define DVSs (see Appendix A.1), however, we need the following construction of DVS-based GL-SPHFs from [BBC+13]. Essentially, a DVS-based GL-SPHF is defined for any algebraic language (see Section 2) $\mathcal{L}_{\mathtt{lpar}}$, where $\mathtt{lpar} = (\boldsymbol{\Gamma}, \boldsymbol{\theta}, \boldsymbol{\lambda})$. Recall that in the case of GL-SPHFs, $\boldsymbol{\Gamma}$ and $\boldsymbol{\theta}$ are affine maps of $\mathtt{x}$, with $\boldsymbol{\Gamma}(\mathtt{x}) \in \mathbb{Z}_p^{n\times k}$, $n > k$. In a DVS-based GL-SPHF for $\mathcal{L}_{\mathtt{lpar}}$, one first samples a hashing key $\mathsf{hk} = \boldsymbol{\alpha} \leftarrow_{\$} \mathbb{Z}_p^n$ and then defines the projection key as $\mathsf{hp} = [\boldsymbol{\gamma}(\mathtt{x})]_1 \leftarrow \mathsf{projkg}(\mathtt{lpar}; \mathsf{hk}) = \boldsymbol{\alpha}^\top [\boldsymbol{\Gamma}]_1 \in \mathbb{G}_1^{1\times k}$. For a witness $\mathtt{w} \in \mathbb{Z}_p^k$, the projection hash is $\mathsf{pH} \leftarrow \mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w}) = [\boldsymbol{\gamma}(\mathtt{x})]_1 \cdot \boldsymbol{\lambda}(\mathtt{x}, \mathtt{w}) = \boldsymbol{\alpha}^\top [\boldsymbol{\Gamma}(\mathtt{x})]_1 \boldsymbol{\lambda}(\mathtt{x}, \mathtt{w}) \in \mathbb{G}_1$. For an input $\mathtt{x} = [\boldsymbol{\theta}]_1 = [\boldsymbol{\Gamma}(\mathtt{x})]_1 \boldsymbol{\lambda}(\mathtt{x}, \mathtt{w}) \in \mathbb{G}_1^n$, the hash is $\mathsf{H} \leftarrow \mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x}) = \mathsf{hk}^\top \cdot \mathtt{x} = \boldsymbol{\alpha}^\top [\boldsymbol{\Gamma}(\mathtt{x})]_1 \boldsymbol{\lambda}(\mathtt{x}, \mathtt{w}) \in \mathbb{G}_1$. Thus, if $\mathtt{x} \in \mathcal{L}_{\mathtt{lpar}}$, then $\mathsf{H} = \mathsf{pH}$. See [BBC+13,Ben16] for the proof of (information-theoretic) smoothness.

4.2 New DVS-Based SZKHF

Recall that a projection key hp is valid if there exists a hk such that $\mathsf{hp} = \mathsf{projkg}(\mathtt{lpar}; \mathsf{hk})$. Consider a DVS-based SPHF HF with $\mathsf{HashSet} = \mathbb{G}_1$ in the

$\mathsf{setup.ltrap}(\mathtt{p})$: return $\mathtt{lpar} \leftarrow_\$ \mathsf{HF.setup.lpar}(\mathtt{p})$;
$\mathsf{hashkg}(\mathtt{lpar})$: return $\mathsf{hk} := \boldsymbol{\alpha} \leftarrow_\$ \mathsf{HF.hashkg}(\mathtt{lpar})$;
$\mathsf{projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$: $\mathsf{hp} := [\boldsymbol{\gamma}]_1 \leftarrow \mathsf{HF.projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$; $\tau \leftarrow_\$ \mathbb{Z}_p^*$; $\mathsf{hp}_{\mathsf{ver}} \leftarrow [\tau, \tau\boldsymbol{\alpha}^\top]_2$;
return $\mathsf{hpf} \leftarrow (\mathsf{hp}, \mathsf{hp}_{\mathsf{ver}})$; // $\mathsf{hp}_{\mathsf{ver}} \in \mathbb{G}_2^{n+1}$;
$\mathsf{verpar}(\mathtt{lpar} = [\boldsymbol{\Gamma}(\mathtt{x})]_1)$: check whether $\mathtt{lpar}$ is well-formed.
$\mathsf{verhp}(\mathtt{lpar} = [\boldsymbol{\Gamma}(\mathtt{x})]_1; \mathsf{hpf} = (\mathsf{hp}, \mathsf{hp}_{\mathsf{ver}}), \mathtt{x})$: check that $[\tau]_2 \in \mathbb{G}_2 \setminus \{1_{\mathbb{G}_2}\}$ and $[\tau\boldsymbol{\alpha}^\top]_2 \bullet [\boldsymbol{\Gamma}(\mathtt{x})]_1 = [\tau]_2 \bullet [\boldsymbol{\gamma}]_1$.
$\mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$: return $\mathsf{HF.hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x}) \bullet [1]_2$;
$\mathsf{projhash}(\mathtt{lpar}; \mathsf{hpf} = (\mathsf{hp}, \mathsf{hp}_{\mathsf{ver}}), \mathtt{x}, \mathtt{w})$: return $\mathsf{HF.projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w}) \bullet [1]_2$;
$\mathsf{simhash}(\mathtt{lpar}; \kappa(\mathsf{hk}) = \kappa(\boldsymbol{\alpha}), \mathtt{x} = [\boldsymbol{\theta}]_1)$: return $\chi(\kappa(\boldsymbol{\alpha}))^\top \cdot [\boldsymbol{\theta}]_1$;

Fig. 4. The GL-SZKHF $\mathsf{HF}_{\mathsf{dvs}}$. Here, HF is any DVS-based SPHF HF with $\mathsf{HashSet} = \mathbb{G}_1$. We denote the procedures of HF by prepending their names with HF as in $\mathsf{HF.hashkg}$. Moreover, $\chi(a) = [a]_2$ if $\kappa = id$ and $\chi(a) = a$ if $\kappa = [\cdot]_2$.

plain model, as defined in Appendix A.1. Since $\boldsymbol{\Gamma}(\mathtt{x}) \in \mathbb{Z}_p^{n \times k}$ with $n > k$, it means that all hp-s are valid. Thus, we must add to the projection key an additional subkey $\mathsf{hp}_{\mathsf{ver}}$ that corresponds to similar auxiliary data $\mathtt{crs}_{\mathsf{CV}}$ in [ABLZ17], such that $\mathsf{hpf} = (\mathsf{hp}, \mathsf{hp}_{\mathsf{ver}})$ fixes uniquely the vector $\boldsymbol{\alpha} \in \mathbb{Z}_p^n$, such that $\mathsf{hp} = \boldsymbol{\alpha}^\top [\boldsymbol{\Gamma}(\mathtt{x})]_1$, and then make it possible to verify that $\mathsf{hp} = \boldsymbol{\alpha}^\top [\boldsymbol{\Gamma}(\mathtt{x})]_1$.

One has to be careful in defining $\mathsf{hp}_{\mathsf{ver}}$. For example, a simple approach is to set $\mathsf{hp}_{\mathsf{ver}} = [\boldsymbol{\alpha}]_2$; after that one can verify hp by just checking that $\mathsf{hp} \bullet [1]_2 = [\boldsymbol{\alpha}]_2^\top \bullet [\boldsymbol{\Gamma}(\mathtt{x})]_1$. Unfortunately, this breaks computational-smoothness, as anybody can check whether $\mathsf{H} = \mathsf{pH}$ by checking whether $\mathsf{H} \bullet [1]_2 = [\boldsymbol{\alpha}]_2^\top \bullet [\boldsymbol{\theta}(\mathtt{x})]_1 = \mathsf{pH} \bullet [1]_2 \in \mathbb{G}_T$. The latter can be done given only $\mathtt{lpar}$, $(\mathsf{hp}, \mathsf{hp}_{\mathsf{ver}})$ and $\mathtt{x}$. To overcome this issue, we use the idea from [BBC+13] to mask $[\boldsymbol{\alpha}]_2$ by multiplying it with a random integer $\tau \in \mathbb{Z}_p^*$. Intuitively, for the construction to be secure, τ has to be chosen so that from

$$\tau(\boldsymbol{\alpha}^\top \boldsymbol{\Gamma}(\mathtt{x}) - \boldsymbol{\gamma}) = \mathbf{0}_{1 \times k} \tag{2}$$

it follows $\boldsymbol{\alpha}^\top \boldsymbol{\Gamma}(\mathtt{x}) = \boldsymbol{\gamma}$. This holds if $\tau \neq 0$. Differently from [BBC+13], we however add the corresponding elements to the projection key (chosen by a potentially malicious verifier) and not to the CRS (chosen by a universally trusted authority).

Moreover, differently from [BBC+13], we allow $\mathtt{lpar}$ to be chosen maliciously. Recall that for this, there must exist an efficient verpar algorithm that verifies that $\mathtt{lpar}$ is well-formed. Such efficient verpar exists only for certain distributions $\mathcal{D}_{\mathsf{p}}$; see [ALSZ20] for discussion. In what follows, we assume that an efficient verpar exists.

The new Sub-ZK SZKHF $\mathsf{HF}_{\mathsf{dvs}}$ with $\mathsf{HashSet} = \mathbb{G}_T$ is depicted in Fig. 4. Next, we define the security assumptions needed to prove its security, and then follow with the security proof. While the construction is inspired by [BBC+13], the security assumptions and the proof are inspired by [ALSZ20].

4.3 New Security Assumptions

In [BBC+13], the DDH adversary $\mathcal{B}$ defined in the computational-smoothness reduction for tsphf relies on the witness-sampleability of $\mathcal{D}_{\mathsf{p}}$ to obtain $([\boldsymbol{\Gamma}(\mathsf{x})]_1, \boldsymbol{\Gamma}(\mathsf{x}))$ sampled from $\mathcal{D}'_{\mathsf{p}}$. Since we prove Sub-PAR smoothness (i.e., smoothness even in the case $[\boldsymbol{\Gamma}(\mathsf{x})]_1$ is maliciously generated), we cannot rely on witness-sampleability. Thus, we need an alternative way to extract $\boldsymbol{\Gamma}(\mathsf{x})$. We follow an idea of [ALSZ20]. Namely, in the proof of Sub-PAR smoothness, $\mathcal{B}$ obtains $[\boldsymbol{\Gamma}(\mathsf{x})]_1 \leftarrow \mathcal{A}(\mathsf{p})$ and then uses a *non-adaptive* discrete logarithm (DL) oracle to extract $\boldsymbol{\Gamma}(\mathsf{x})$. Hence, instead of the DDH assumption (together with witness-sampleability) that was used in [BBC+13], we prove (non-blackbox) Sub-PAR smoothness under the following new non-falsifiable, non-adaptive interactive $\mathsf{DDH}^{\mathsf{dl}}$ assumption.

The $\mathsf{DDH}^{\mathsf{dl}}$ assumption is an non-adaptive X^Y-type interactive assumption, where the assumption X is assumed to hold even if the adversary is given a non-adaptive (i.e., before the challenge X is chosen), access to an oracle that solves the assumption Y. Several X^Y assumptions are known in the literature, see, e.g., [Gjø06,Lip10,ALSZ20]. Some X^Y assumptions (e.g., the ones used in [Lip10]) are falsifiable; however, $\mathsf{DDH}^{\mathsf{dl}}$ is non-falsifiable.

Let $\iota \in \{1,2\}$. The $\mathsf{DDH}^{\mathsf{dl}}_{\mathbb{G}_\iota}$ assumption states that DDH in $\mathbb{G}_\iota$ remains intractable even if the adversary is given a non-adaptive access to the DL oracle. More precisely, *the* $\mathsf{DDH}^{\mathsf{dl}}_{\mathbb{G}_\iota}$ *assumption* holds relative to Pgen, if $\forall$ PPT $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda); st \leftarrow \mathcal{A}^{\mathsf{dl}(\cdot)}(\mathsf{p}); x,y,z \leftarrow_{\$} \mathbb{Z}_p; w_0 \leftarrow xy; w_1 \leftarrow z;\\ b \leftarrow_{\$} \{0,1\}; b^* \leftarrow \mathcal{A}(\mathsf{p}, st, [x,y,w_b]_\iota) : \ b = b^*\end{array}\right] \approx_\lambda 0 \ .$$

New knowledge assumptions. Let $\mathsf{HF} = \mathsf{HF}_{\mathsf{dvs}}$ be the new SZKHF. To prove ZK and persistent ZK properties in our construction, we need to rely on two new assumptions $\mathsf{X\text{-}SZKHF\text{-}KE}$, for $X \in \{\mathcal{D}_{\mathsf{p}}, \mathsf{SUBPAR}\}$. We first define these assumptions. In Theorem 2, we prove their security in the AGM [FKL18]. The knowledge assumptions are to postulate that given a valid hpf, one can efficiently extract $\mathtt{td} = \kappa(\mathsf{hk})$. More precisely, $\mathsf{SUBPAR\text{-}SZKHF\text{-}KE}$ (resp., $\mathcal{D}_{\mathsf{p}}\text{-}\mathsf{SZKHF\text{-}KE}$) assumption is the core of the persistent ZK proof (resp., the ZK proof) of the DVS-based SZKHF construction in Theorem 3. There, we assume that if an adversary $\mathcal{A}$ outputs a language parameter $\mathtt{lpar}$ accepted by verpar and a hpf accepted by verhp, then there exists an extractor $\mathsf{Ext}_{\mathcal{A}}$ that by knowing the secret coins of $\mathcal{A}$, returns $\mathtt{td} = \kappa(\mathsf{hk})$ where hk was used to compute hpf.

Like KWKE [ALSZ20] is a tautological knowledge assumption for the Kiltz-Wee QA-NIZK [KW15], $\mathsf{X\text{-}SZKHF\text{-}KE}$ is tautological knowledge assumption for $\mathsf{HF}_{\mathsf{dvs}}$. Nevertheless, KWKE has already found uses behind its original application in [ALSZ20], and we hope the same will happen to $\mathsf{X\text{-}SZKHF\text{-}KE}$.

Definition 3. *Let κ be a one-to-one map. Fix $n > k \geq 1$ and a distribution $\mathcal{D}_{\mathsf{p}}$. Let $\mathsf{HF} = \mathsf{HF}_{\mathsf{dvs}}$ be the new GL-SZKHF. The* $\mathsf{X\text{-}SZKHF\text{-}KE}$ *assumption for $X \in \{\mathcal{D}_{\mathsf{p}}, \mathsf{SUBPAR}\}$ holds relative to* Pgen *for any $\mathsf{p} \in \mathrm{im}(\mathsf{Pgen}(1^\lambda))$ and PPT adversary $\mathcal{A}$ and PPT subverter $\mathcal{Z}$, there exists a PPT extractor $\mathsf{Ext}_{\mathcal{Z}}$, such that $\Pr[\mathsf{Exp}^{\mathsf{X\text{-}SZKHF\text{-}KE}}_{\mathsf{HF},\mathcal{Z},\mathsf{Ext}_{\mathcal{Z}},\mathcal{A}}(\lambda) = 1] \approx_\lambda 0$, where $\mathsf{Exp}^{\mathsf{X\text{-}SZKHF\text{-}KE}}_{\mathsf{HF},\mathcal{Z},\mathsf{Ext}_{\mathcal{Z}},\mathcal{A}}(\lambda)$ is depicted in Fig. 5.*

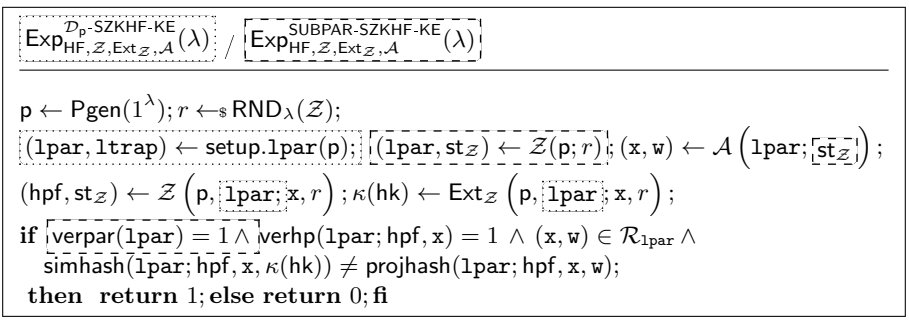

Fig. 5. SZKHF-KE experiments in Definition 3. The dotted-boxed / dashed-box part is only present in dotted-boxed / dashed-boxed experiments.

Theorem 2 (Security of X-SZKHF-KE). *Let κ be a one-to-one map. Fix $n > k \geq 1$. Then* SUBPAR-SZKHF-KE *and* $\mathcal{D}_\mathsf{p}$-SZKHF-KE *hold relative to* Pgen *in the AGM.*

The proof of Theorem 2 is given in Appendix E.2. If $\kappa(\mathsf{hk}) = [\mathsf{hk}]_2$ then it suffices to extract $[\mathsf{hk}]_2$. Then, one can rewrite the proof so that the algebraic adversary only recovers the coefficients of $\tau(\boldsymbol{Q}_2)$ but not of $\boldsymbol{\Delta}(\boldsymbol{Q}_2)$. In that case, one can prove persistent ZK and Sub-ZK (see Theorem 3) under standard knowledge assumptions (instead relying on the AGM) by adding $[y\tau]_2$ to hp_ver, where $y \leftarrow\!\!\$\, \mathbb{Z}_p$ is a knowledge trapdoor (i.e., only adding one additional group element to the projection key). Alternatively, one can define new tautological knowledge assumptions stating that given hpf as input, one can extract either hk or $\kappa(\mathsf{hk})$.

4.4 Security Proof

Theorem 3. *Let $\{\mathcal{L}_\mathtt{lpar}\}$ be a family of algebraic languages, such that there exists an efficient* verpar *algorithm. Let* HF *be a DVS-based GL-SPHF for $\{\mathcal{L}_\mathtt{lpar}\}$ and let HF_dvs be the GL-SZKHF for $\{\mathcal{L}_\mathtt{lpar}\}$ depicted in Fig. 4.*

(i) If $\mathsf{DDH}^\mathsf{dl}_{\mathbb{G}_2}$ holds relative to Pgen*, then HF_dvs is a (non-blackbox) Sub-PAR computationally-smooth GL-SZKHF in the plain model.*

(ii) Let $\kappa := a \mapsto [a]_2$ or $\kappa := id$. The GL-SZKHF HF_dvs is (a) auxiliary-string non-blackbox persistent ZK under SUBPAR-SZKHF-KE*, and (b) no-auxiliary-string non-blackbox ZK under $\mathcal{D}_\mathsf{p}$-SZKHF-KE, in the plain model.*

Proof. **(i: Sub-PAR smoothness).** First, recall that computational Sub-PAR (non-blackbox) smoothness says that for all PPT adversaries $\mathcal{A}$, $\Pr[\mathsf{Smooth}^{\mathsf{nbb\text{-}plain}}_{\mathsf{HF},\mathcal{A}}(\lambda) = 1] \approx_\lambda 1/2$, where the experiment $\mathsf{Smooth}^{\mathsf{nbb\text{-}plain}}_{\mathsf{HF},\mathcal{A}}(\lambda)$ is depicted in Fig. 6. (Compared to Fig. 2 in Section 3, we just changed the general procedures with their instantiations in Fig. 11.)

We first reduce the Sub-PAR smoothness to the following intermediate assumption: for all $\mathsf{p} \in \mathrm{im}(\mathsf{Pgen}(1^\lambda))$, and stateful PPT adversaries $\mathcal{B}$,

$\mathsf{Smooth}^{\mathsf{nbb\text{-}plain}}_{\mathsf{HF},\mathcal{A}}(\lambda)$

$\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda)$;
$(\mathtt{lpar} = [\boldsymbol{\Gamma}]_1, \mathtt{x}) \leftarrow \mathcal{A}(\mathsf{p})$;
$(\mathsf{hpf} = (\mathsf{hp}, \mathsf{hp}_{\mathsf{ver}}), \mathsf{hk} = \boldsymbol{\alpha}) \leftarrow \mathsf{kgen}([\boldsymbol{\Gamma}]_1; \mathtt{x})$;
$\mathsf{H}_0 \leftarrow \mathsf{hash}([\boldsymbol{\Gamma}]_1; \boldsymbol{\alpha}, \mathtt{x})$; $\mathsf{H}_1 \leftarrow\!\!\$\ \mathsf{HashSet}$; $b \leftarrow\!\!\$\ \{0,1\}$; $b' \leftarrow \mathcal{A}(\mathsf{hpf}, \mathsf{H}_b)$;
if $\mathsf{verpar}(\mathtt{lpar}) = 1 \ \wedge \ b' = b \wedge \neg(\exists \mathtt{w} : \mathtt{x} = [\boldsymbol{\Gamma}]_1 \mathtt{w})$
then return 1; **else return** 0; **fi**

Fig. 6. Experiment $\mathsf{Smooth}^{\mathsf{nbb\text{-}plain}}_{\mathsf{HF},\mathcal{A}}(\lambda)$

$\mathsf{Exp}^{\mathsf{int}}_{\mathcal{B}}(\mathsf{p})$

$[\boldsymbol{\Gamma}]_1 \leftarrow \mathcal{B}(\mathsf{p})$;
$\boldsymbol{\alpha} \leftarrow\!\!\$\ \mathbb{Z}_p^n$; // $\mathsf{hk} \leftarrow \mathsf{HF.hashkg}(\mathtt{lpar})$;
$[\boldsymbol{\gamma}]_1 \leftarrow \boldsymbol{\alpha}^\top [\boldsymbol{\Gamma}]_1$; // $\mathsf{hp} \leftarrow \mathsf{HF.projkg}(\mathtt{lpar}; \mathsf{hk})$;
$\tau \leftarrow\!\!\$\ \mathbb{Z}_p^*$; $\mathsf{hp}_{\mathsf{ver}} \leftarrow [\tau, \tau\boldsymbol{\alpha}^\top]_2$; $\mathsf{hpf} \leftarrow (\mathsf{hp}, \mathsf{hp}_{\mathsf{ver}})$;
$b \leftarrow\!\!\$\ \{0,1\}$;
if $b = 0$ **then** $\boldsymbol{\beta} \leftarrow \boldsymbol{\alpha}$; **else** $\boldsymbol{\mu} \leftarrow\!\!\$\ \mathrm{coker}(\boldsymbol{\Gamma})$; $\boldsymbol{\beta} \leftarrow \boldsymbol{\alpha} + \boldsymbol{\mu}$;
// $\boldsymbol{\beta}$ is either sk or $\mathsf{sk}+$ random element of the cokernel
$b' \leftarrow \mathcal{B}(\mathsf{hpf}, [\boldsymbol{\beta}]_2)$;
return $b = b'$;

Fig. 7. Experiment $\mathsf{Exp}^{\mathsf{int}}_{\mathcal{B}}(\mathsf{p})$ for the proof of Sub-PAR smoothness in Theorem 5

$\Pr[\mathsf{Exp}^{\mathsf{int}}_{\mathcal{B}}(\mathsf{p}) = 1] \approx_\lambda 1/2$, where $\mathsf{Exp}^{\mathsf{int}}_{\mathcal{B}}(\mathsf{p})$ is depicted in Fig. 7. Intuitively, this assumption states that for any PPT adversary (who is given the projection key hpf), it is hard to distinguish $[\mathsf{hk}]_2$ from $[\mathsf{hk}]_2 + \boldsymbol{\mu}$, where $\boldsymbol{\mu}$ is a random element of the cokernel of $[\boldsymbol{\Gamma}]_2$. That is, hpf does not contain sufficient information to decide which of the possible $|\mathrm{coker}(\boldsymbol{\Gamma})|$ secret keys was used by the challenger. Note that since $\boldsymbol{\mu} \in \mathrm{coker}(\boldsymbol{\Gamma})$, $(\boldsymbol{\alpha} + \boldsymbol{\mu})^\top \boldsymbol{\Gamma} = \boldsymbol{\alpha}^\top \boldsymbol{\Gamma}$.

Let $\mathcal{A}$ be a Sub-PAR smoothness adversary. We construct an adversary $\mathcal{B}$ against the intermediate problem that uses the help of $\mathcal{A}$. The idea is to let $\mathcal{B}$ play the role of the challenger in the smoothness experiment and feed $\mathcal{A}$ with values calculated based on the intermediate experiment. $\mathcal{B}(\mathsf{p})$ proceeds as follows:

1. $([\boldsymbol{\Gamma}]_1, \mathtt{x}) \leftarrow \mathcal{A}(\mathsf{p})$.
2. Return $[\boldsymbol{\Gamma}]_1$ to the challenger.
3. The challenger creates $(\mathsf{hpf}, [\boldsymbol{\beta}]_2)$ as in Fig. 7, and sends it to $\mathcal{B}$.
4. $\mathcal{B}$ computes $\mathsf{H}_b \leftarrow [\boldsymbol{\beta}]_2^\top \mathtt{x}$.
5. $b' \leftarrow \mathcal{A}(\mathsf{hpf}, \mathsf{H}_b)$.
6. Return b'.

Clearly, if $b = 0$ (i.e., $\boldsymbol{\beta} = \boldsymbol{\alpha}$), then $\mathsf{H}_b = \mathsf{H}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$. Otherwise (i.e., if $b = 1$), we have two cases:

- if $\mathtt{x} \notin \mathcal{L}_{\mathtt{lpar}}$, then $\mathsf{H}_b = [\boldsymbol{\alpha}+\boldsymbol{\mu}]_2^\top \mathtt{x} = [\boldsymbol{\alpha}]_2^\top \mathtt{x} + [\boldsymbol{\mu}]_2^\top \mathtt{x}$ is uniformly random from the viewpoint of the adversary. This is because in this case, $\mathtt{x}$ is not in the column span of $[\boldsymbol{\Gamma}]_1$ and thus $\mathsf{H} = [\boldsymbol{\alpha}]_2^\top \mathtt{x}$ is uniformly random.
- if $\mathtt{x} \in \mathcal{L}_{\mathtt{lpar}}$, then $\mathsf{H}_b = [\boldsymbol{\beta}]_2^\top \mathtt{x} = [\boldsymbol{\alpha}]_2^\top \mathtt{x} = \mathsf{H}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$.

Now assume that $\mathcal{A}$ breaks the Sub-PAR smoothness with non-negligible advantage. This means that with non-negligible probability, $\mathcal{A}$ outputs $b = 0$ in the case of receiving a real hash and outputs $b = 1$ in the case of receiving a random hash. Based on the above observation, this would be the same as the advantage of $\mathcal{B}$ in succeeding in $\mathsf{Exp}^{\mathsf{int}}_{\mathcal{B}}(\mathsf{p})$.

We now show that the intermediate assumption can be reduced to the $\mathsf{DDH}^{\mathsf{dl}}_{\mathbb{G}_2}$ problem. Let $\mathcal{D}$ be an adversary against the $\mathsf{DDH}^{\mathsf{dl}}_{\mathbb{G}_2}$ problem. Without loss of generality, we assume that the challenge given to $\mathcal{D}$ is of the form $[x, xy, z]_2$, where $x, y, z \in \mathbb{Z}_p$ and $z = y$ or random [8]. $\mathcal{D}$ plays the role of the challenger for $\mathcal{B}$ in the experiment $\mathsf{Exp}^{\mathsf{int}}_{\mathcal{B}}(\mathsf{p})$ in Fig. 7. Before describing the reduction, note that for all $[\boldsymbol{\gamma}]_1 \in \{[\boldsymbol{\gamma}']_1 \in \mathbb{G}_1^{1\times k} : \exists \boldsymbol{\alpha} \in \mathbb{Z}_p^n \text{ s.t. } \boldsymbol{\gamma}' = \boldsymbol{\alpha}^\top \boldsymbol{\Gamma}\} \subseteq \mathbb{G}_1^{1\times k}$, there exists $\Delta_{\boldsymbol{\gamma}} \in \mathbb{Z}_p^{n\times(m+1)}$, with $m = n - k$, such that

$$\{\boldsymbol{\alpha} : \boldsymbol{\gamma} = \boldsymbol{\alpha}^\top \boldsymbol{\Gamma}\} = \{\Delta_{\boldsymbol{\gamma}} \cdot \tilde{\boldsymbol{s}} : \tilde{\boldsymbol{s}} = \left(\begin{smallmatrix} \boldsymbol{s} \\ 1 \end{smallmatrix}\right); \forall \boldsymbol{s} \in \mathbb{Z}_p^m\} \ .$$

In other words, the columns of $\Delta_{\boldsymbol{\gamma}}$ form a basis for the solutions of equation $\boldsymbol{\gamma} = \boldsymbol{\alpha}^\top \boldsymbol{\Gamma}$ with unknown $\boldsymbol{\alpha}$[9]. By having this, the adversary $\mathcal{D}$ plays the role of the challenger for $\mathcal{B}$ as follows:

1. run $\mathcal{B}$ with input p and obtain $[\boldsymbol{\Gamma}]_1$.
2. call the DL oracle on input $[\boldsymbol{\Gamma}]_1$ and set $st := \boldsymbol{\Gamma} \in \mathbb{Z}_p^{n\times k}$.
3. given a challenge $C = [x, xy, z]_2$, generate m DDH challenges $\boldsymbol{C} = \{[x, r_i xy, r_i z]_2\}_{i\in[m]}$ for $r_i \leftarrow_{\$} \mathbb{Z}_p$ by using the self-randomizability of the DDH problem. To simplify the notation, we write $\boldsymbol{C} = [x, x\boldsymbol{y}, \boldsymbol{z}]_2$.
4. call $\mathcal{B}$ with input $([\tau, \boldsymbol{F}, \boldsymbol{G}]_2, [\boldsymbol{H}]_1)$ defined as $\tau = x$, $[\boldsymbol{F}]_2 = \Delta_{\boldsymbol{\gamma}} \cdot x\tilde{\boldsymbol{y}} = x(\Delta_{\boldsymbol{\gamma}} \cdot \tilde{\boldsymbol{y}})$, $[\boldsymbol{G}]_2 = \Delta_{\boldsymbol{\gamma}} \cdot \tilde{\boldsymbol{z}}$ and $[\boldsymbol{H}]_1 = [\boldsymbol{\gamma}]_1$, where $\boldsymbol{\gamma} \leftarrow_{\$} \mathbb{Z}_p^k$.
5. return $\mathcal{B}$'s output.

Note that when $\boldsymbol{C} = [x, x\boldsymbol{y}, \boldsymbol{z}]_2$ is a vector of DDH tuples, then $\boldsymbol{z} = \boldsymbol{y}$ and $\mathcal{B}$ is given $([\tau, \tau\boldsymbol{\alpha}^\top, \boldsymbol{\alpha}]_2, \boldsymbol{\alpha}^\top [\boldsymbol{\Gamma}]_1)$ as input, where $\boldsymbol{\alpha} = \Delta_{\boldsymbol{\gamma}} \cdot \tilde{\boldsymbol{y}}$. Thus $\mathcal{B}$ is expected to output $b' = 0$. On the other hand, if $\boldsymbol{C} = [x, x\boldsymbol{y}, \boldsymbol{z}]_2$ is not a vector of DDH tuples, then $\boldsymbol{z}$ is a random vector different from $\boldsymbol{y}$, but still such that $\Delta_{\boldsymbol{\gamma}} \cdot \tilde{\boldsymbol{z}} \in \{\boldsymbol{\alpha} : \boldsymbol{\gamma} = \boldsymbol{\alpha}^\top \boldsymbol{\Gamma}\}$. This means that in this case, $\mathcal{B}$ is given $([\tau, \tau\boldsymbol{\alpha}^\top, \boldsymbol{\beta}]_2, \boldsymbol{\alpha}^\top [\boldsymbol{\Gamma}]_1)$ as input, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are random vectors that are solutions for $\{\boldsymbol{\alpha} : \boldsymbol{\gamma} = \boldsymbol{\alpha}^\top \boldsymbol{\Gamma}\}$. This is $\mathcal{B}$'s input in Fig. 7 experiment for the case $b = 1$ and therefore, $\mathcal{B}$ is expected to output $b' = 1$. This completes the proof of Sub-PAR smoothness.

(ii-a: persistent ZK). Let $\mathcal{Z}$ be a subverter that breaks the Sub-ZK property. First, $\mathcal{Z}(\mathsf{p}; r_{\mathcal{Z}})$ outputs $([\boldsymbol{\Gamma}]_1, \mathsf{aux}_{\mathsf{hp}})$. Let $\mathcal{B}$ be the adversary from Fig. 8. Note that $\mathsf{RND}_\lambda(\mathcal{B}) = \mathsf{RND}_\lambda(\mathcal{Z})$. Under the $\mathsf{SUBPAR\text{-}SZKHF\text{-}KE}$ assumption, there exists an extractor $\mathsf{Ext}^2_{\mathcal{B}}$, such that if $\mathsf{verpar}([\boldsymbol{\Gamma}]_1) = 1$ and

[8] Although this tuple is different from the usual DDH challenge $[x, y, z]_2$ where $z = xy$ or random, it is not hard to show they are two versions of the same hardness problem.

[9] The existence of $\Delta_{\boldsymbol{\gamma}}$ comes from the parametric equations that describe all the solutions of the underlying system of equations.

$\mathsf{verhp}([\boldsymbol{\Gamma}]_1, \mathsf{hpf}) = 1$ then $\mathsf{Ext}^2_{\mathcal{B}}(\mathsf{p}; r_{\mathcal{Z}})$ outputs $\boldsymbol{\alpha}$, such that $\boldsymbol{\gamma} = \boldsymbol{\alpha}^\top \boldsymbol{\Gamma}$. We construct a trivial extractor $\mathsf{Ext}_{\mathcal{Z}}(\mathsf{p}; r_{\mathcal{Z}})$ for $\mathcal{Z}$, as depicted in Fig. 8. Clearly, $\mathsf{Ext}_{\mathcal{Z}}$ returns $\mathsf{hk} = \boldsymbol{\alpha}$, such that $\boldsymbol{\gamma} = \boldsymbol{\alpha}^\top \boldsymbol{\Gamma}$.

$\mathcal{B}(\mathsf{p}; r_{\mathcal{Z}})$	$\mathsf{Ext}_{\mathcal{Z}}(\mathsf{p}; r_{\mathcal{Z}})$
$([\boldsymbol{\Gamma}]_1, \mathsf{hpf}, \mathsf{st}_{\mathcal{Z}}) \leftarrow \mathcal{Z}(\mathsf{p}; r_{\mathcal{Z}})$; **return** hpf;	**return** $\mathsf{Ext}^2_{\mathcal{B}}(\mathsf{p}; r_{\mathcal{Z}})$;

Fig. 8. The extractor and the constructed adversary $\mathcal{B}$ from the persistent zero-knowledge proof of Theorem 3.

Fix concrete values of λ and $r_{\mathcal{Z}} \in \mathsf{RND}_\lambda(\mathcal{Z})$. Let $\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda)$, $(\mathtt{lpar} = [\boldsymbol{\Gamma}]_1, \mathsf{st}_{\mathcal{Z}}) \leftarrow \mathcal{Z}(\mathsf{p}; r_{\mathcal{Z}})$, $(\mathtt{x} = [\boldsymbol{y}]_1, \mathtt{w} = \boldsymbol{w}) \leftarrow \mathcal{A}(\mathtt{lpar}; \mathsf{st}_{\mathcal{Z}})$, $(\mathsf{hpf}, \mathsf{st}_{\mathcal{Z}}) \leftarrow \mathcal{Z}(\mathsf{p}; \mathtt{x}; r_{\mathcal{Z}})$, and run $\mathsf{Ext}_{\mathcal{Z}}(\mathsf{p}; r_{\mathcal{Z}})$ to obtain $\boldsymbol{\alpha}$.

It clearly suffices to show that if $\mathsf{verpar}(\mathtt{lpar}) = 1$, $\mathsf{verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 1$ and $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$, then $\mathsf{projhash}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}, \mathtt{w}) = [\boldsymbol{\gamma}]_1 \boldsymbol{w} \bullet [1]_2$ and $\mathsf{simhash}(\mathtt{lpar}, \kappa(\mathsf{hk}) = \kappa(\boldsymbol{\alpha}); \mathsf{hp}, \mathtt{x}) = \chi(\kappa(\boldsymbol{\alpha}))^\top \mathtt{x}$ (for χ defined in Fig. 4) have the same distribution. Really, from $\mathsf{verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 1$ it follows $\boldsymbol{\gamma} = \boldsymbol{\alpha}^\top \boldsymbol{\Gamma}$ and from $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$ it follows $\mathtt{x} = \boldsymbol{\Gamma} \boldsymbol{w}$. Thus, $\mathsf{projhash}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}, \mathtt{w}) = [\boldsymbol{\gamma}]_1 \boldsymbol{w} \bullet [1]_2 = [\boldsymbol{\alpha}^\top \boldsymbol{\Gamma} \boldsymbol{w}]_1 \bullet [1]_2 = [\boldsymbol{\alpha}]_2^\top \mathtt{x} = \mathsf{simhash}(\mathtt{lpar}, \kappa(\mathsf{hk})$. Hence, projhash and simhash have the same distribution, and thus, $\mathsf{HF}_{\mathsf{dvs}}$ is persistent zero-knowledge under SUBPAR-SZKHF-KE assumption.

(ii-b: ZK). The proof can directly be captured from the proof in (ii-a) when $[\boldsymbol{\Gamma}]_1$ is picked honestly and $\mathcal{Z}$ gets it as an additional input. □

References

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. doi:10.1007/978-3-319-70700-6_1.

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. doi:10.1007/978-3-662-46803-6_3.

ALSZ20. Behzad Abdolmaleki, Helger Lipmaa, Janno Siim, and Michal Zajac. On QA-NIZK in the BPK model. In *PKC 2020, Part I*, LNCS, pages 590–620. Springer, Heidelberg, 2020. doi:10.1007/978-3-030-45374-9_20.

APV05. Joël Alwen, Giuseppe Persiano, and Ivan Visconti. Impossibility and feasibility results for zero knowledge with public keys. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 135–151. Springer, Heidelberg, August 2005. doi:10.1007/11535218_9.

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. doi:10.1007/978-3-642-40041-4_25.

BCKL08. Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 356–374. Springer, Heidelberg, March 2008. doi:10.1007/978-3-540-78524-8_20.

BCNP04. Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th FOCS*, pages 186–195. IEEE Computer Society Press, October 2004. doi:10.1109/FOCS.2004.71.

Ben16. Fabrice Ben Hamouda-Guichoux. *Diverse Modules and Zero-Knowledge*. PhD thesis, PSL Research University, 2016.

BFM88. Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. doi:10.1145/62212.62222.

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. doi:10.1007/978-3-662-53890-6_26.

CGGM00. Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *32nd ACM STOC*, pages 235–244. ACM Press, May 2000. doi:10.1145/335305.335334.

CH20. Geoffroy Couteau and Dominik Hartmann. Shorter non-interactive zero-knowledge arguments and ZAPs for algebraic languages. LNCS, pages 768–798. Springer, Heidelberg, 2020. doi:10.1007/978-3-030-56877-1_27.

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. doi:10.1007/3-540-46035-7_4.

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. doi:10.1007/978-3-642-40084-1_8.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. LNCS, pages 33–62. Springer, Heidelberg, 2018. doi:10.1007/978-3-319-96881-0_2.

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. doi:10.1007/978-3-319-76578-5_11.

GGSW13. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013. doi:10.1145/2488608.2488667.

Gjø06. Kristian Gjøsteen. A new security proof for damgård's ElGamal. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 150–158. Springer, Heidelberg, February 2006. doi:10.1007/11605805_10.

GL03. Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, Heidelberg, May 2003. http://eprint.iacr.org/2003/032.ps.gz. doi:10.1007/3-540-39200-9_33.

GO94. Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994. doi:10.1007/BF00195207.

GPS08. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for Cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

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. doi:10.1007/978-3-642-42033-7_1.

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. doi:10.1007/978-3-642-19571-6_18.

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. doi:10.1007/978-3-662-46803-6_4.

Lip10. Helger Lipmaa. On the CCA1-Security of Elgamal and Damgård's Elgamal. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Inscrypt 2010*, volume 6584 of *LNCS*, pages 18–35, Shanghai, China, October 20–23, 2010. Springer, Heidelberg. doi:https://doi.org/10.1007/978-3-642-21518-6_2.

Lip19. Helger Lipmaa. Simulation-Extractable ZK-SNARKs Revisited. Technical Report 2019/612, IACR, May 31, 2019. https://eprint.iacr.org/2019/612, updated on 8 Feb 2020.

MR01. Silvio Micali and Leonid Reyzin. Soundness in the public-key model. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 542–565. Springer, Heidelberg, August 2001. doi:10.1007/3-540-44647-8_32.

Wee07. Hoeteck Wee. Lower bounds for non-interactive zero-knowledge. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 103–117. Springer, Heidelberg, February 2007. doi:10.1007/978-3-540-70936-7_6.

A Additional Preliminaries

A.1 Diverse Vector Space (DVS).

A DVS [BBC+13,ABP15,Ben16] is essentially a representation of a language $\mathcal{L} \subseteq \mathcal{X}$ as a subspace $\hat{\mathcal{L}}$ of some vector space. Let $\mathcal{R}_{\mathcal{L}} = \{(\mathtt{x}, \mathtt{w})\}$ be a relation with $\mathcal{L} = \{\mathtt{x} : \exists \mathtt{w}, (\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathcal{L}}\}$. Let p be system parameters, including say the description of a bilinear group. Let $\boldsymbol{\Gamma}_{\mathtt{lpar}}(\mathtt{x})$ be an $n \times k$ matrix, $\boldsymbol{\theta}_{\mathtt{lpar}}(\mathtt{x})$ an n-dimensional vector, and $\boldsymbol{\lambda}_{\mathtt{lpar}}(\mathtt{x}, \mathtt{w})$ a k-dimensional vector. A (pairing-based) DVS $\mathcal{V}$ is equal to $\mathcal{V} = (\mathsf{p}, \mathcal{X}, \mathcal{L}, \mathcal{R}_{\mathcal{L}}, n, k, \boldsymbol{\Gamma}, \boldsymbol{\theta}, \boldsymbol{\lambda})$. The matrix $\boldsymbol{\Gamma}_{\mathtt{lpar}}(\mathtt{x})$ can depend on $\mathtt{x}$ (in this case, we say that we have a GL-DVS) or not (KV-DVS). Moreover, different coefficients of $\boldsymbol{\theta}_{\mathtt{lpar}}(\mathtt{x})$, $\boldsymbol{\Gamma}_{\mathtt{lpar}}(\mathtt{x})$, and $\boldsymbol{\lambda}_{\mathtt{lpar}}(\mathtt{x}, \mathtt{w})$ can belong to different algebraic structures (most commonly, given a bilinear group $\mathsf{p} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e})$, either to $\mathbb{Z}_p$, $\mathbb{G}_1$, $\mathbb{G}_2$, or $\mathbb{G}_T$) as long as the equation $\boldsymbol{\theta}_{\mathtt{lpar}}(\mathtt{x}) = \boldsymbol{\Gamma}_{\mathtt{lpar}}(\mathtt{x}) \cdot \boldsymbol{\lambda}_{\mathtt{lpar}}(\mathtt{x}, \mathtt{w})$ is "well-typed". That is, an equation like $\left(\begin{smallmatrix} [\theta_1]_T \\ [\theta_2]_1 \end{smallmatrix}\right) = \left(\begin{smallmatrix} [\Gamma_{11}]_T & [\Gamma_{12}]_2 \\ [\Gamma_{21}]_1 & \Gamma_{22} \end{smallmatrix}\right)\left(\begin{smallmatrix} \lambda_1 \\ [\lambda_2]_1 \end{smallmatrix}\right)$ holds. Note that $\mathcal{L}_{\mathtt{lpar}} = \{\mathtt{x} : \exists \boldsymbol{\lambda}, \boldsymbol{\theta}_{\mathtt{lpar}}(\mathtt{x}) = \boldsymbol{\Gamma}_{\mathtt{lpar}}(\mathtt{x}) \cdot \boldsymbol{\lambda}\}$. We omit the subscript $\mathtt{lpar}$ if it is clear from the context.

A DVS $\mathcal{V}$ satisfies the following properties [Ben16]: (i) *coordinate-independence of groups:* the group in which each coordinate of $\boldsymbol{\theta}(\mathtt{x})$ lies is independent of $\mathtt{x}$. (ii) *perfect completeness:* for any $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathcal{L}}$, $\boldsymbol{\theta}(\mathtt{x}) = \boldsymbol{\Gamma}(\mathtt{x}) \cdot \boldsymbol{\lambda}(\mathtt{x}, \mathtt{w})$. (iii) *statistical ε-soundness:* $\forall \mathtt{x} \in \mathcal{X}_{\mathtt{lpar}} \setminus \mathcal{L}_{\mathtt{lpar}}$, $\Pr[\boldsymbol{\theta}(\mathtt{x}) \in \mathrm{colspace}(\boldsymbol{\Gamma}(\mathtt{x}))] \leq \varepsilon$.

Construction of SPHF from DVS. Given a GL/KV-DVS for $\mathcal{L}$, one can construct an efficient GL/KV-SPHF for $\mathtt{x}' \in \mathcal{L}$, where $\mathtt{w} = \boldsymbol{\lambda}(\mathtt{x}', \mathtt{w}')$ and $\mathtt{x} = [\boldsymbol{\theta}(\mathtt{x}')]_\iota = [\boldsymbol{\Gamma}(\mathtt{x}')]_\iota \boldsymbol{\lambda}(\mathtt{x}', \mathtt{w}')$ [BBC+13], see Fig. 9. Here, the only possible nonlinear operation is the dependency of $\boldsymbol{\theta}$ and $\boldsymbol{\Gamma}$ on the actual input $\mathtt{x}'$. It is known that if $\mathcal{V}$ is a 0-sound GL-DVS/KV-DVS, then the PHF in Fig. 9 is a 0-smooth GL-SPHF/KV-SPHF, see Theorem 3.1.11 in [Ben16].

Example 1. Let $\mathcal{L} := \langle [\mathsf{v}]_1 \rangle$ where $[\mathsf{v}]_1 \in \mathbb{G}_1^{2\times 1}$. Since $\mathcal{L}$ is a subspace, $\mathcal{L} = \hat{\mathcal{L}}$. Thus, $[\boldsymbol{\theta}(\mathtt{x})]_1 = \mathtt{x} \in \mathbb{G}_1^2$ (thus $\mathtt{x} = \mathtt{x}'$) and $[\boldsymbol{\Gamma}(\mathtt{x})]_1 = [\mathsf{v}]_1 \in \mathbb{G}_1^{2\times 1}$. Clearly, $[\boldsymbol{\theta}(\mathtt{x})]_1 \in \langle [\mathsf{v}]_1 \rangle$ iff $[\boldsymbol{\theta}(\mathtt{x})]_1 = [\boldsymbol{\Gamma}(\mathtt{x})]_1 \cdot \lambda(\mathtt{x}, \mathtt{w}) = [\mathsf{v}]_1 \lambda$ for some $\lambda = \lambda(\mathtt{x}, \mathtt{w}) \in \mathbb{Z}_p$. Now, $\mathsf{hk} = (\alpha_1, \alpha_2)^\top \leftarrow_{\$} \mathbb{Z}_p^2$, $\mathsf{hp} = \alpha_1[\mathsf{v}_1]_1 + \alpha_2[\mathsf{v}_2]_1$, $\mathsf{pH} = \mathsf{hp} \cdot \lambda = \lambda \cdot (\alpha_1[\mathsf{v}_1]_1 + \alpha_2[\mathsf{v}_2]_1) = \mathsf{hk} \cdot [\boldsymbol{\theta}(\mathtt{x})]_1 = \mathsf{H}$.

One can construct a DVS for any subspace language $\mathcal{L}$ by letting $\boldsymbol{\theta}(\mathtt{x}) = \mathtt{x}$ to be the identity map and $\boldsymbol{\Gamma} = \boldsymbol{\Gamma}(\mathtt{x})$ to be the basis matrix of $\mathcal{L}$.

$\mathsf{hashkg}(\mathtt{lpar})$: sample $\boldsymbol{\alpha} \leftarrow_{\$} \mathbb{Z}_p^n$, and output $\mathsf{hk} \leftarrow \boldsymbol{\alpha}$;
$\mathsf{projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x} = [\boldsymbol{\theta}(\mathtt{x}')]_\iota)$: $[\gamma]_\iota^\top \leftarrow \boldsymbol{\alpha}^\top [\boldsymbol{\Gamma}(\mathtt{x}')]_\iota \in \mathbb{G}_\iota^{1\times k}$; return $\mathsf{hp} \leftarrow [\gamma]_\iota$;
$\mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$: return $\mathsf{H} \leftarrow \boldsymbol{\alpha}^\top [\boldsymbol{\theta}(\mathtt{x}')]_\iota$;
$\mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w} = \boldsymbol{\lambda}(\mathtt{x}', \mathtt{w}'))$: return $\mathsf{pH} \leftarrow [\gamma]_\iota^\top \boldsymbol{\lambda}(\mathtt{x}', \mathtt{w}')$;

Fig. 9. DVS-based SPHF. Here, $\mathtt{lpar} = [\boldsymbol{\Gamma}(\mathtt{x}')]_\iota$, $\mathsf{hk} = \boldsymbol{\alpha}$, $\mathsf{hp} = [\gamma]_\iota$, and $\mathtt{x} = [\boldsymbol{\theta}(\mathtt{x}')]_\iota$.

A.2 Algebraic Group Model (AGM)

AGM is a new model [FKL18] used to prove the security of a cryptographic assumption, protocol, or a primitive. Essentially, in the AGM, one assumes that each PPT algorithm $\mathcal{A}$ is algebraic in the following sense. Assume $\mathcal{A}$'s input includes $[\boldsymbol{x}_\iota]_\iota$ and no other elements from the group $\mathbb{G}_\iota$. Moreover, assume $\mathcal{A}$ has an access to an oracle O, such that $\mathsf{O}(\iota)$ samples and outputs a random element $[q_{\iota k}]_\iota$ from $\mathbb{G}_\iota$, $\iota \in \{1,2\}$. The oracle access models the ability of $\mathcal{A}$ to create random group elements. We assume that if $\mathcal{A}$ outputs group elements $[\boldsymbol{y}_\iota]_\iota$, then $\mathcal{A}$ knows matrices $\boldsymbol{N}_\iota$, such that $\boldsymbol{y}_\iota = \boldsymbol{N}_\iota(\begin{smallmatrix}\boldsymbol{x}_\iota\\ \boldsymbol{q}_\iota\end{smallmatrix})$.

More precisely, a PPT algorithm $\mathcal{A}$ is *(*Pgen*-)algebraic* if there exists an efficient extractor $\mathsf{Ext}_{\mathcal{A}}$, s.t. for any PPT sampleable distribution $\mathcal{D}$, $\mathsf{Adv}^{\mathrm{agm}}_{\mathsf{Pgen},\mathcal{D},\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(\lambda) :=$

$$\Pr\left[\begin{array}{l}\mathsf{p} \leftarrow_{\$} \mathsf{Pgen}(1^\lambda); \mathsf{x} = ([\boldsymbol{x}_1]_1, [\boldsymbol{x}_2]_2) \leftarrow_{\$} \mathcal{D}; r \leftarrow_{\$} \mathsf{RND}_\lambda(\mathcal{A});\\ ([\boldsymbol{y}_1]_1, [\boldsymbol{y}_2]_2) \leftarrow_{\$} \mathcal{A}^{\mathsf{O}}(\mathsf{x}; r); (\boldsymbol{N}_1, \boldsymbol{N}_2) \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{x}; r) :\\ (\boldsymbol{y}_1 \neq \boldsymbol{N}_1(\begin{smallmatrix}\boldsymbol{x}_1\\ \boldsymbol{q}_1\end{smallmatrix}) \vee \boldsymbol{y}_2 \neq \boldsymbol{N}_2(\begin{smallmatrix}\boldsymbol{x}_2\\ \boldsymbol{q}_2\end{smallmatrix}))\end{array}\right] = \mathsf{negl}(\lambda) \ .$$

O is an oracle, that given $\iota \in \{1,2\}$ as an input, samples and returns a random element from $\mathbb{G}_\iota$, and $[\boldsymbol{q}_\iota]_\iota$ is the list of all elements output by $\mathsf{O}(\iota)$. The AGM states that $\mathsf{Adv}^{\mathrm{agm}}_{\mathsf{Pgen},\mathcal{D},\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(\lambda) = \mathsf{negl}(\lambda)$ for any PPT sampleable $\mathcal{D}$ and PPT $\mathcal{A}$.

A.3 Bare Public Key Model

In the Bare Public Key (BPK) model [CGGM00,MR01], parties have access to a public file F, a polynomial-size collection of records (id, pk_{id}), where id is a string identifying a party (e.g., a verifier), and pk_{id} is her (alleged) public key. In a typical zero-knowledge protocol in the BPK model, a key-owning party $\mathcal{P}_{id}$ works in two stages. In stage one (the *key-generation stage*), on input a security parameter 1^λ and randomizer r, $\mathcal{P}_{id}$ outputs a public key pk_{id} and stores the corresponding secret key sk_{id}. We assume the *no-auxiliary-string BPK* model where from this it follows that $\mathcal{P}_{id}$ actually created pk_{id}. After that, F will include (id, pk_{id}). In stage two, each party has access to F, while $\mathcal{P}_{id}$ has possibly access to sk_{id} (however, the latter is not required by us). It is commonly assumed that only the verifier of a NIZK argument system in the BPK model has a public key [MR01].

A.4 Registered Public Key Model.

In the registered public key (RPK, [BCNP04]) model, each party $\mathcal{P}_{id}$ trusts *some* key-registration authority $\mathcal{R}_{id}$ and has registered her key with $\mathcal{R}_{id}$. The same $\mathcal{R}_{id}$ can be used by several parties, or each party can choose to trust a separate authority. If $\mathcal{P}_{id}$ is honest, then the secret key sk exists and the public key pk comes from correct distribution (in this case, the public key is said to

be *"safe"*). If $\mathcal{P}_{id}$ is dishonest, sk still exists and pk has been computed from it honestly, but there is no guarantee about its distribution (in this case, pk is said to be *"well-formed"*). Different variants (most importantly, the "traditional proof-of-knowledge" version where sk and the public key are generated by $\mathcal{P}_{id}$ who then sends pk to $\mathcal{R}_{id}$ and proves the knowledge of sk to $\mathcal{R}_{id}$ by using a stand-alone zero-knowledge proof) of the RPK model are known. Each party knows the identities of all other parties and their key-registration authorities, see [BCNP04] for discussion. We describe the bare public key (BPK) model in Appendix A.3. The RPK model is significantly weaker than the CRS model but it can be interpreted as the CRS model in the special case when the RPK creator is universally trusted. The BPK model is even weaker than the RPK model.

B Blackbox SZKHF in the RPK Model

Since blackbox SZKHFs are impossible in the plain model, we will next consider blackbox SZKHFs in the RPK model [BCNP04]. The following definition combines the security definitions of Sub-PAR QA-NIZKs in the BPK model [ALSZ20] with these of TSPHFs [BBC+13]. We will first give the new definition and then explain the difference between the new definition and the definitions of [ALSZ20] and [BBC+13].

A SZKHF in the RPK model is defined together with new algorithms $\mathsf{K}_{\mathsf{rpk}}$, verpar, verhp and $\mathsf{simhash}$ as follows.

- Pgen, $\mathsf{setup.ltrap}$ are as before, except that $\mathsf{setup.ltrap}$ obeys the rules of the RPK model. (See the description of $\mathsf{K}_{\mathsf{rpk}}$ below.)
- $\mathsf{verpar}(\mathtt{lpar})$: outputs 1 if $\mathtt{lpar}$ is well-formed and 0 otherwise.
- $\mathsf{K}_{\mathsf{rpk}}(\mathtt{lpar})$: takes an input $\mathtt{lpar}$ generated by $\mathsf{setup.ltrap}$ and outputs a public key $\mathtt{rpk}$ together with a secret key sk. $\mathsf{K}_{\mathsf{rpk}}$ can either generate sk herself or can, alternatively, verify that the owner of sk knows the secret key corresponding to $\mathtt{rpk}$, [BCNP04]. In the latter case, $\mathsf{K}_{\mathsf{rpk}}$ can be implemented as a stand-alone interactive zero-knowledge protocol where a party registers her public key $\mathtt{rpk}$ with an authority by additionally proving the knowledge of $\mathtt{td} := \mathsf{sk}$. In a security proof, $\mathtt{td}$ is then extracted by using (say) rewinding. If the creator of $\mathtt{rpk}$ is untrusted, $\mathtt{rpk}$ is well-formed and its underlying sk is returned; however, there is no guarantee about the distribution of $\mathtt{rpk}$ or sk. The $\mathsf{setup.ltrap}$ algorithm works similarly, but with $\mathtt{rpk}$ being replaced with $\mathtt{lpar}$ and sk being replaced with $\mathtt{ltrap}$. Thus, if the creator of $\mathtt{lpar}$ is untrusted, $\mathtt{lpar}$ is well-formed and its underlying $\mathtt{ltrap}$ is returned; however, there is no guarantee about the distribution of $\mathtt{lpar}$ or $\mathtt{ltrap}$.
- hashkg, projkg, hash, and $\mathsf{projhash}$ are as usual but also take $\mathtt{rpk}$ as an input. To shorten notation, we will denote "$\mathsf{hk} \leftarrow \mathsf{hashkg}(\mathtt{lpar}, \mathtt{rpk}); \mathsf{hp} \leftarrow \mathsf{projkg}(\mathtt{lpar}, \mathtt{rpk}; \mathsf{hk}, \mathtt{x})$" by "$(\mathsf{hp}, \mathsf{hk}) \leftarrow \mathsf{kgen}(\mathtt{lpar}, \mathtt{rpk}; \mathtt{x})$".
- $\mathsf{verhp}(\mathtt{lpar}, \mathtt{rpk}; \mathsf{hp}, \mathtt{x})$: outputs 1 if hp is a valid projection key and 0 otherwise.
- $\mathsf{simhash}(\mathtt{lpar}; \mathtt{td}, \mathsf{hp}, \mathtt{x})$: returns the simulated (trapdoor) hash value of $\mathtt{x}$, given an RPK trapdoor $\mathtt{td}$ and hp.

In applications where $\mathsf{K}_{\mathsf{rpk}}$ is not trusted (like the definition of zero-knowledge in the RPK model), we denote the untrusted $\mathsf{K}_{\mathsf{rpk}}$ as $\mathsf{K}^{\mathsf{adv}}_{\mathsf{rpk}}$. As above, the output of $\mathsf{K}^{\mathsf{adv}}_{\mathsf{rpk}}$ will be well-formed (in particular, it will return a correct sk) but there will be no assumption about the distribution of $\mathtt{rpk}$ and $\mathsf{K}^{\mathsf{adv}}_{\mathsf{rpk}}$ may leak information about sk to other adversaries.

Definition 4. *A blackbox GL-SZKHF* $\mathsf{HF} = (\mathsf{Pgen}, \mathsf{setup.ltrap}, \mathsf{hashkg}, \mathsf{projkg}, \mathsf{hash}, \mathsf{projhash})$ *in the RPK model must satisfy the following properties for some PPT* $\mathsf{K}_{\mathsf{rpk}}$*, deterministic polynomial-time* verpar*,* verhp *and* $\mathsf{simhash}$*, and the experiments depicted in Figs. 1 to 3.*

Perfect completeness: *for any* λ *and PPT* $\mathcal{A}$*,* $\Pr[\mathsf{Complete}^{\mathsf{rpk}}_{\mathsf{HF},\mathcal{A}}(\lambda) = 1] = 1$.

Computational (blackbox) smoothness: *for any PPT* $\mathsf{K}^{\mathsf{adv}}_{\mathsf{rpk}}$ *and* $\mathcal{A}$*,* $\Pr[\mathsf{Smooth}^{\mathsf{bb\text{-}rpk}}_{\mathsf{HF},\mathcal{A}}(\lambda) = 1] \approx_\lambda \frac{1}{2}$*. SZKHF is* statistically smooth *if the same holds for all unbounded adversaries.*

Composable (blackbox) persistent ZK in the RPK model: *For any PPT* $\mathcal{Z}$ *and unbounded* $\mathcal{A}$*,* $\Pr[\mathsf{PZK}^{\mathsf{bb\text{-}rpk}}_{\mathsf{HF},\mathcal{Z},\mathcal{A}}(\lambda) = 1] \approx_\lambda \frac{1}{2}$.

Comparison with previous work. Differently from [ALSZ20], the definitions are for SPHFs and not for QA-NIZKs. Our definition is in the RPK model for a trusted public key, without a non-blackbox extractor of the secret key.

Moreover, since we want to avoid non-blackbox techniques, in the definition of smoothness, we assume that also $\mathtt{lpar}$ is generated according to the rules of the RPK model (that is, $\mathsf{setup.ltrap}$ returns $\mathtt{lpar}$ with a corresponding $\mathtt{ltrap}$). This is motivated by the fact that existing TSPHF constructions [BBC+13] are given for witness-sampleable distributions, where $\mathtt{ltrap}$ is used in the smoothness proofs explicitly. We modify the way the witness-sampleable distribution is used according to the model. In the RPK model, the RPK-model setup algorithm returns $\mathtt{ltrap}$. In the non-blackbox plain model of Section 3.2, we will assume the existence of an extractor that can extract $\mathtt{ltrap}$. In both cases, $\mathtt{ltrap}$ will be used in the smoothness and persistent ZK security proofs, and we do not assume that $\mathtt{ltrap}$ is correctly distributed. As in [ALSZ20], persistent zero-knowledge means zero-knowledge in the case when $\mathtt{lpar}$ is maliciously constructed.

On the other hand, [BBC+13] defined TSPHFs in the CRS model (where there exists a universally trusted third party that creates a CRS), while we use the significantly weaker RPK model. More importantly, we consider the case of maliciously created $\mathtt{lpar}$; this seems to be a first in the existing SPHF literature. More precisely, we assume that both $\mathtt{lpar}$ and $\mathtt{rpk}$ are constructed according to the rules of the RPK model. We only considered honest $\mathtt{lpar}$ in Definition 1 since there we gave an impossibility result. In the RPK model, we are interested in a possibility result; thus, following [ALSZ20], we consider persistent ZK.[10] We also use a language that immediately guarantees composability of SZKHFs.

[10] We emphasize that proving ZK in the case of subverted $\mathtt{lpar}$ and hp is paramount in applications where both $\mathtt{lpar}$ and hp are generated by the verifier (the party who checks that the values of hash and $\mathsf{projhash}$ are equal).

Finally, [BBC+13] used different terminology: what we call zero-knowledge was called soundness in [BBC+13]; however, it was called zero-knowledge in [Ben16].

Abdolmaleki *et al.* [ALSZ20] showed that in the case of QA-NIZKs, while ZK (with honestly chosen $\mathtt{lpar}$) sounds to be a weaker definition than persistent ZK (with maliciously chosen $\mathtt{lpar}$), this is actually not the case. More precisely, they constructed a contrived QA-NIZK argument system Π_{leaky} where one need $\mathtt{ltrap}$ to be able to simulate. In the case of persistent ZK, one can use a knowledge extractor (the use of which is explicitly allowed by their definition of persistent ZK) to obtain $\mathtt{ltrap}$ and then use $\mathtt{ltrap}$ to simulate. However, Π_{leaky} does not achieve ZK since a simulator does not have access to $\mathtt{ltrap}$. In our definition of persistent ZK in the RPK model, there is no extractor and thus ZK follows from the persistent ZK. However, we will use an extractor in Section 3.2 and thus there we will define ZK and persistent ZK separately.

Finally, we emphasize that persistent ZK holds in the case the RPK is honestly created (and thus $\mathsf{simhash}$ has access to the secret key $\mathtt{td}$) but $\mathtt{lpar}$ and hp are subverted. Thus, like TSPHFs, SZKHFs in the RPK model provide only partial answer to the problem of subversion. To solve the latter, in Section 3.2, we define Sub-ZK SZKHF (in the plain model).

Constructions. We give a construction of computationally-smooth blackbox SZKHF in the RPK model from $\mathsf{HF}_{\mathsf{dvs}}$ in Fig. 4, by defining $\mathtt{rpk} = [\tau]_2$ and $\mathsf{hpf} = (\mathsf{hp}, \mathsf{hp}_{\mathsf{ver}})$ for $\mathsf{hp}_{\mathsf{ver}} = [\tau\boldsymbol{\alpha}^\top]_2$, such that the Eq. (2) holds. Similar to Sub-ZK SZKHF, in the blackbox SZKHF we need $\tau \neq 0$ for the computational-smoothness and persistent ZK properties.

Theorem 4. *(i) If $\tau \in \mathbb{Z}_p^*$, $[\boldsymbol{\Gamma}]_1$ is witness-sampleable, and* DDH *holds relative to* Pgen*, then $\mathsf{HF}_{\mathsf{dvs}}$ is computationally smooth in the RPK model.*

(ii) Let $\kappa := a \mapsto [a]_2$ or $\kappa := id$. Assuming $\tau \in \mathbb{Z}_p^*$, $\mathsf{HF}_{\mathsf{dvs}}$ is persistent ZK in the RPK model.

Proof. **(i: computational smoothness).** Similar to the proof of computational smoothness in Theorem 3, except that instead of using the DL oracle, we now use witness-sampleability to recover $\boldsymbol{\Gamma}$.

(ii: persistent ZK). The subverter $\mathcal{Z}$ gets public parameters p as input, and outputs a language parameter $\mathtt{lpar} = [\boldsymbol{\Gamma}]_1$ and auxiliary state $\mathsf{st}_{\mathcal{Z}}$. Then given $[\boldsymbol{\Gamma}]_1$, the (malicious) $\mathsf{K}_{\mathsf{rpk}}^{\mathsf{adv}}$ algorithm generates $(\mathtt{rpk} = [\tau]_2, \mathtt{td} = \tau)$. Due to the properties of the RPK model, $\mathtt{rpk}$ is a valid public key corresponding to the secret key sk. Let $\mathcal{A}$ be an adversary against persistent ZK and given auxiliary state $\mathsf{st}_{\mathcal{Z}}$ and $\mathtt{rpk}$, outputs a word $\mathtt{x}$. The subverter $\mathcal{Z}(\mathsf{p}, \mathtt{x}, \mathtt{rpk})$ outputs $\mathsf{hpf} = (\mathsf{hp} = \boldsymbol{\alpha}^\top[\boldsymbol{\Gamma}]_1, \mathsf{hp}_{\mathsf{ver}} = [\tau\boldsymbol{\alpha}^\top]_2)$ and some auxiliary state $\mathsf{st}_{\mathcal{Z}}$. Then (blackbox) persistent ZK follows directly from the fact that one can run $\mathsf{simhash}([\boldsymbol{\Gamma}]_1, \tau; \mathsf{hpf}, \mathtt{x})$ and simulate H with following: (i) by using τ, extract $[\boldsymbol{\alpha}^\top]_2$ form $[\tau\boldsymbol{\alpha}^\top]_2$ and (ii) compute $\mathsf{H} \leftarrow [\boldsymbol{\alpha}^\top]_2\mathtt{x}$. □

We also give another construction of computationally-smooth blackbox SZKHF in the RPK model based on TSPHF [BBC+13] in Appendix C). Note that both constructions are computationally-smooth SZKHF under the DDH assumption for witness-sampleable languages.

C SZKHF Constructions based of TSPHF

In this section, we show how to construct a Sub-ZK SZKHF in the plain model (resp. (blackbox) SZKHF in the RPK model) from TSPHF construction of [BBC+13] (see Appendix C), where all hp, $\mathtt{crs}$ and the language parameter $\mathtt{lpar}$ can be created by the hp generator in the plain model (resp. both hp and $\mathtt{lpar}$ in the RPK model). We first recall the definition of TSPHF.

TSPHF construction of [BBC+13]. Benhamouda *et al.* [BBC+13] proposed the following trapdoor SPHF (TSPHF) tsphf, based on any SPHF with $\mathsf{HashSet} = \mathbb{G}_1$. See Fig. 10. Note that $\mathsf{H}^* = \mathtt{td}^{-1} \cdot \mathtt{x}^\top[\boldsymbol{\zeta}]_2 = \mathtt{td}^{-1}\mathtt{x}^\top[\boldsymbol{\alpha} \cdot \mathtt{td}]_2 = \mathtt{x}^\top[\boldsymbol{\alpha}]_2 = \boldsymbol{\alpha}^\top \cdot \mathtt{x} \bullet [1]_2 = \mathsf{H} \in \mathbb{G}_T$. If $\mathsf{verhp}(\mathtt{lpar}, \mathtt{crs}; \mathsf{hpf}, \mathtt{x})$ accepts then $\mathsf{hp} = [\boldsymbol{\Gamma}]_1^\top(\mathtt{td}^{-1} \cdot \boldsymbol{\zeta}) = [\boldsymbol{\Gamma}]_1^\top \boldsymbol{\alpha}'$ for $\mathsf{hk}' = \boldsymbol{\alpha}' := \mathtt{td}^{-1} \cdot \boldsymbol{\zeta}$ and thus $\mathsf{hp} = \mathsf{HF.projkg}(\mathtt{lpar}; \mathsf{hk}', \mathtt{x})$.

$\mathsf{setup.lpar}(\mathsf{p})$: return $\mathsf{HF.setup.lpar}(\mathsf{p})$;
$\mathsf{K_{crs}}(\mathtt{lpar})$: $\mathtt{td} \leftarrow_\$ \mathbb{Z}_p^*$; $\mathtt{crs} \leftarrow [\mathtt{td}]_2$; return $(\mathtt{crs}, \mathtt{td})$;
$\mathsf{hashkg}(\mathtt{lpar})$: return $\mathsf{hk} := \boldsymbol{\alpha} \leftarrow \mathsf{HF.hashkg}(\mathtt{lpar})$; // $\mathsf{hk} \in \mathbb{Z}_p^n$
$\mathsf{projkg}(\mathtt{lpar}, \mathtt{crs}; \mathsf{hk}, \mathtt{x})$: $\mathsf{hp} \leftarrow \mathsf{HF.projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$; $[\boldsymbol{\zeta}]_2 \leftarrow \boldsymbol{\alpha} \cdot [\mathtt{td}]_2 \in \mathbb{G}_2^n$; return $\mathsf{hpf} = (\mathsf{hp}, [\boldsymbol{\zeta}]_2)$;
$\mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$: return $\mathsf{HF.hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x}) \bullet [1]_2$;
$\mathsf{projhash}(\mathtt{lpar}, \mathtt{crs}; \mathsf{hp}, \mathtt{x}, \mathtt{w})$: return $\mathsf{HF.projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w}) \bullet [1]_2$;
$\mathsf{simhash}(\mathtt{lpar}, \mathtt{td}; \mathsf{hp}, \mathtt{x})$: return $\mathsf{H}^* \leftarrow \mathtt{td}^{-1} \cdot \mathtt{x}^\top[\boldsymbol{\zeta}]_2$.
$\mathsf{verhp}(\mathtt{lpar} = [\boldsymbol{\Gamma}]_1, \mathtt{crs} = [\mathtt{td}]_2; \mathsf{hpf} = (\mathsf{hp}, [\boldsymbol{\zeta}]_2), \mathtt{x})$: checks whether $\mathtt{crs} \neq [0]_2$, $[\boldsymbol{\zeta}]_2 \in \mathbb{G}_2^n$, and $\mathsf{hp} \bullet [\mathtt{td}]_2 = [\boldsymbol{\Gamma}]_1^\top \bullet [\boldsymbol{\zeta}]_2 (\in \mathbb{G}_T^k)$.

Fig. 10. The Benhamouda *et al.*TSPHF tsphf [BBC+13]. Here, HF is any DVS-based SPHF HF. We denote the procedures of HF by prepending their names with HF as in $\mathsf{HF.hashkg}$.

For this TSPHF construction to work, HF must not use pairings and $\mathtt{lpar}$, $\mathtt{x}$, and hp must belong to source groups; this is since one needs to pair $\mathtt{x}$ and $[\boldsymbol{\zeta}]_2$, hp and $[\mathtt{td}]_2$, and $[\boldsymbol{\Gamma}]_1$ and $[\boldsymbol{\zeta}]_2$. Benhamouda *et al.* [BBC+13] proved that tsphf is computationally-smooth under the DDH assumption in $\mathbb{G}_2$. tsphf was defined in the CRS model, but it is easy to see that it stays secure also in the RPK model, given a honestly generated $\mathtt{lpar}$.

TSPHF $\Rightarrow$ Sub-ZK SZKHF. One can see the DVS-based SZKHF construction in 4.2 as the TSPHF of Benhamouda-Pointcheval. But (i) in the former, the language parameter $\mathtt{lpar}$ can be subverted, and (ii) in the latter, $\mathtt{crs}$ contains $[\tau]_2$ for $\tau \leftarrow_\$ \mathbb{Z}_p^*$ while in Sub-ZK SZKHFs in plain model, $[\tau]_2$ is constructed as part of the projection key. In Fig. 11, we give a generic conversion from TSPHF to Sub-ZK SZKHF in the plain model.

In Theorem 5, we prove $\mathsf{HF}_{\mathsf{tsphf}}$ in Fig. 11 is computationally Sub-PAR smooth under the $\mathsf{DDH}^{\mathsf{dl}}$ assumption and persistent ZK and ZK under the $\mathsf{SUBPAR\text{-}SZKHF\text{-}KE}$ and $\mathcal{D}_{\mathsf{p}}\text{-}\mathsf{SZKHF\text{-}KE}$ assumptions respectively.

$\mathsf{setup}(\mathsf{p})$: return $\mathsf{tsphf.setup}(\mathsf{p})$;
$\mathsf{hashkg}(\mathtt{lpar})$: return $\mathsf{tsphf.hashkg}(\mathtt{lpar})$;
$\mathsf{projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$: $(\mathtt{crs}, \mathtt{td}) \leftarrow_{\$} \mathsf{tsphf.K}_{\mathtt{crs}}(\mathtt{lpar})$; $(\mathsf{hp}, \mathsf{hp}_{\mathsf{ver}}) \leftarrow \mathsf{tsphf.projkg}(\mathtt{lpar}, \mathtt{crs}; \mathsf{hk}, \mathtt{x})$; ${\mathsf{hp}_{\mathsf{ver}}}^* \leftarrow (\mathtt{crs}, \mathsf{hp}_{\mathsf{ver}})$; return $\mathsf{hpf} \leftarrow (\mathsf{hp}, {\mathsf{hp}_{\mathsf{ver}}}^*)$;
$\mathsf{verpar}(\mathtt{lpar} = [\boldsymbol{\Gamma}]_1)$: checks whether $\mathtt{lpar}$ is well-formed.
$\mathsf{verhp}(\mathtt{lpar} = [\boldsymbol{\Gamma}]_1; \mathsf{hpf}, \mathtt{x})$: checks whether $\mathsf{hpf} = (\mathsf{hp}, {\mathsf{hp}_{\mathsf{ver}}}^*)$, such that $\mathsf{tsphf.verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 1$.
// $[\boldsymbol{\zeta}]_2 \in \mathbb{G}_2^n$, $\mathtt{crs} \neq 0$,
// and $\mathtt{hp}[\mathtt{td}]_2 = [\boldsymbol{\Gamma}]_1^\top [\boldsymbol{\zeta}]_2 (\in \mathbb{G}_T^k)$.
$\mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$: return $\mathsf{tsphf.hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$;
$\mathsf{projhash}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}, \mathtt{w})$: write $\mathsf{hpf} = (\mathtt{crs}, \mathsf{hp}_{\mathsf{ver}}, \mathsf{hp})$; return $\mathsf{tsphf.projhash}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}, \mathtt{w})$;
$\mathsf{simhash}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{td}, \mathtt{x})$: return $\mathsf{tsphf.hash}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{td}, \mathtt{x})$;

Fig. 11. $\mathsf{HF}_{\mathsf{tsphf}}$

Theorem 5. *Let $\kappa := a \mapsto [a]_2$ or $\kappa := id$. Then $\mathsf{HF}_{\mathsf{tsphf}}$ in Fig. 11 is (i) computationally Sub-PAR smooth under the $\mathsf{DDH}^{\mathsf{dl}}$ assumption, and (ii) κ-extractable (ii-a) persistent-ZK under* SUBPAR-SZKHF-KE *assumption, and (ii-b) ZK under $\mathcal{D}_{\mathsf{p}}$-SZKHF-KE assumption in the plain model.*

Proof. **(i: Sub-PAR smoothness).** The proof is similar to the smoothness proof of Theorem 3.

(ii-a: persistent ZK). The proof can be captured from the persistent ZK proof of Theorem 3. Given $r \leftarrow_{\$} \mathsf{RND}_\lambda(\mathcal{Z})$, the subverter $\mathcal{Z}$ first pick $\mathtt{lpar}$ together with some auxiliary information $\mathsf{st}_{\mathcal{Z}}$. Then after receiving $\mathtt{x} \leftarrow \mathcal{A}(\mathsf{st}_{\mathcal{Z}})$, the subverter $\mathcal{Z}$ with input the word $\mathtt{x}$ and the random tape r, outputs hpf and some auxiliary state $\mathsf{st}_{\mathcal{Z}}$. Let $\mathsf{RND}_\lambda(\mathcal{A}) = \mathsf{RND}_\lambda(\mathcal{Z})$. Under the SUBPAR-SZKHF-KE assumption, there exists an extractor $\mathsf{Ext}_{\mathcal{Z}}$, such that if $\mathsf{verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 1$ and $\mathsf{verpar}(\mathtt{lpar}) = 1$ then $\mathsf{Ext}_{\mathcal{Z}}(\mathsf{p}; \mathtt{x}, r)$ outputs $\kappa(\mathsf{hk})$.

Fix $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$, λ, $\mathsf{p} \in \mathrm{im}(\mathsf{Pgen}(1^\lambda))$, $r \in \mathsf{RND}_\lambda(\mathcal{Z})$, and run $\mathsf{Ext}_{\mathcal{Z}}(\mathsf{p}; \mathtt{x}, r)$ to obtain $\kappa(\mathsf{hk})$. It clearly suffices to show that if $\mathsf{verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 1$ and $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}_{\mathtt{lpar}}$ then

$$\mathsf{O}_0(\mathtt{x}, \mathtt{w}) = \mathsf{projhash}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}, \mathtt{w}) = \mathsf{pH} \ ,$$
$$\mathsf{O}_1(\mathtt{x}, \mathtt{w}) = \mathsf{simhash}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}, \kappa(\mathsf{hk})) = \mathsf{sH}$$

have the same distribution. This holds since from $\mathsf{verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 1$ it follows $\mathsf{O}_0(\mathtt{x}, \mathtt{w}) = \mathsf{pH} = \mathsf{sH} = \mathsf{O}_1(\mathtt{x}, \mathtt{w})$. Hence, O_0 and O_1 have the same distribution.

(ii-b: ZK). The proof can be captured from the ZK proof of Theorem 4. The subverter $\mathcal{Z}$ gets as an input the language parameter $\mathtt{lpar}$ and a random tape r, and outputs hpf and some auxiliary state $\mathsf{st}_{\mathcal{Z}}$. Let $\mathsf{RND}_\lambda(\mathcal{A}) = \mathsf{RND}_\lambda(\mathcal{Z})$. Under the $\mathcal{D}_{\mathsf{p}}$-SZKHF-KE assumption, there exists an extractor $\mathsf{Ext}_{\mathcal{Z}}$, such that if $\mathsf{verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 1$ then $\mathsf{Ext}_{\mathcal{Z}}(\mathsf{p}, \mathtt{lpar}, \mathtt{x}; r)$ outputs $\kappa(\mathsf{hk})$.

Fix $\mathtt{lpar} \leftarrow_{\$} \mathcal{D}_{\mathsf{p}}$, $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$, $r \in \mathsf{RND}_\lambda(\mathcal{Z})$, and run $\mathsf{Ext}_{\mathcal{Z}}(\mathsf{p}, \mathtt{lpar}; \mathtt{x}, r)$ to obtain $\kappa(\mathsf{hk})$. It clearly suffices to show that if $\mathsf{verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 1$ and

$(\mathtt{x},\mathtt{w}) \notin \mathcal{R}_{\mathtt{lpar}}$ then

$$\begin{aligned}\mathsf{O}_0(\mathtt{x},\mathtt{w}) &= \mathsf{projhash}(\mathtt{lpar};\mathsf{hpf},\mathtt{x},\mathtt{w}) = \mathsf{pH}\ ,\\ \mathsf{O}_1(\mathtt{x},\mathtt{w}) &= \mathsf{simhash}(\mathtt{lpar};\mathsf{hpf},\mathtt{x},\kappa(\mathsf{hk})) = \mathsf{sH}\end{aligned}$$

have the same distribution. This holds since from $\mathsf{verhp}(\mathtt{lpar};\mathsf{hpf},\mathtt{x}) = 1$ it follows $\mathsf{O}_0(\mathtt{x},\mathtt{w}) = \mathsf{pH} = \mathsf{sH} = \mathsf{O}_1(\mathtt{x},\mathtt{w})$. Hence, O_0 and O_1 have the same distribution.

TSPHF $\Rightarrow$ (blackbox) SZKHF in the RPK model. The generic conversion from TSPHF construction of [BBC+13] to (blackbox) SZKHF in the RPK model is straight forward by setting $\mathtt{rpk} := [\tau]_2$ and $\tau \leftarrow_{\$} \mathbb{Z}_p \setminus \{0\}$. In Theorem 6, we prove the TSPHF-based construction of (blackbox) SZKHF is computationally smooth under the DDH assumption and persistent ZK in the RPK model.

Theorem 6. *(i) If $\tau \in \mathbb{Z}_p \setminus \{0\}$, $[\boldsymbol{\Gamma}]_1$ is witness-sampleable, and* DDH *holds relative to* Pgen*, then the TSPHF-based construction of (blackbox) SZKHF is computationally smooth in the RPK model.*

(ii) Let $\kappa := a \mapsto [a]_2$ or $\kappa := id$. Assuming $\tau \in \mathbb{Z}_p \setminus \{0\}$, the TSPHF-based construction of (blackbox) SZKHF is persistent ZK in the RPK model.

Proof. **(i: Smoothness).** The proof is similar to the smoothness proof of Theorem 4.

(ii: persistent ZK). As in the RPK model, the trapdoor τ is honestly generated by a trusted party so the proof is similar to the persistent ZK of Theorem 5. □

D Sub-ZK NIZKs

We recall the definition of (Sub-ZK) NIZKs in the BPK model from [ABLZ17,Fuc18,ALSZ20]. A Sub-ZK NIZK system Π for a relation $\mathcal{R}$ consists of five PPT algorithms in the BPK model:

$\mathsf{K}_{\mathsf{bpk}}(\mathtt{p})$**:** given $\mathtt{p}$ outputs a trapdoor $\mathtt{td}$ and a public key $\mathtt{bpk}$. Otherwise, it outputs $\perp$.

$\mathsf{PKV}(\mathtt{bpk})$**:** given a public key $\mathtt{bpk}$, returns either 0 (reject) or 1 (accept) if it is well-formed.

$\mathsf{P}(\mathtt{bpk},\mathtt{x},\mathtt{w})$**:** given a public key $\mathtt{bpk}$, a statement $\mathtt{x}$, and a witness $\mathtt{w}$, outputs an argument π if $(\mathtt{x},\mathtt{w}) \in \mathcal{R}$. Otherwise, it outputs $\perp$.

$\mathsf{V}(\mathtt{bpk},\pi,\mathtt{x})$**:** given a public key $\mathtt{bpk}$, a statement $\mathtt{x}$, and a proof π,, returns either 0 (reject) or 1 (accept).

$\mathsf{Sim}(\mathtt{bpk},\mathtt{x},\mathtt{td})$**:** given a public key $\mathtt{bpk}$, a statement $\mathtt{x}$, and a trapdoor $\mathtt{td}$ outputs an argument π. Otherwise, it outputs $\perp$.

Here, $\mathsf{K}_{\mathsf{bpk}}$ is the public key generation algorithm, PKV is the public-key verification algorithm, P is the prover, V is the verifier, and Sim is the simulator

$\mathsf{Complete}^{\mathtt{bpk}}_{\Pi,\mathcal{A}}(\lambda)$

$\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda); (\mathtt{bpk}, \mathtt{td}) \leftarrow \mathsf{K}_{\mathtt{bpk}}(\mathsf{p});\ (\mathtt{x}, \mathtt{w}) \leftarrow \mathcal{A}(\mathsf{p}, \mathtt{bpk});$
if $\mathsf{PKV}(\mathtt{bpk}) = 1 \wedge \big((\mathtt{x}, \mathtt{w}) \notin \mathcal{R} \vee \mathsf{V}(\mathtt{bpk}, \pi, \mathtt{x})\big)$
then return 1; **else return** 0; **fi**

Fig. 12. Completeness experiment in Definition 5.

$\mathsf{ZK}^{\mathtt{bpk}}_{\Pi,\mathcal{Z},\mathcal{A}}(\lambda)$

$\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda); r \leftarrow\!\!\$\ \mathsf{RND}_\lambda(\mathcal{Z}); (\mathtt{bpk}, \mathtt{st}_{\mathcal{Z}}) \leftarrow \mathcal{Z}(\mathsf{p}; r); (\mathtt{x}, \mathtt{w}) \leftarrow \mathcal{A}(\mathtt{bpk}; \mathtt{st}_{\mathcal{Z}});$
$\mathtt{td} \leftarrow \mathsf{Ext}_{\mathcal{Z}}(\mathsf{p}; r); \pi_0 \leftarrow \mathsf{P}(\mathtt{bpk}, \mathtt{x}, \mathtt{w}); \pi_1 \leftarrow \mathsf{Sim}(\mathtt{bpk}, \mathtt{x}, \mathtt{td});$
$b \leftarrow\!\!\$\ \{0,1\}; b' \leftarrow \mathcal{A}(\mathsf{p}, \mathtt{st}_{\mathcal{Z}}, \mathtt{bpk}, \pi_b);$ **if** $(\mathtt{x}, \mathtt{w}) \in \mathcal{R} \wedge \mathsf{PKV}(\mathtt{bpk}) = 1 \wedge b' = b;$
then return 1; **else return** 0; **fi**

Fig. 13. Sub-ZK experiment in Definition 5.

$\mathsf{Snd}^{\mathtt{bpk}}_{\Pi,\mathcal{A}}(\lambda)$

$\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda); (\mathtt{bpk}, \mathtt{td}) \leftarrow \mathsf{K}_{\mathtt{bpk}}(\mathsf{p}); (\pi, \mathtt{x}) \leftarrow \mathcal{A}(\mathsf{p}, \mathtt{bpk});$
if $\mathsf{V}(\mathtt{bpk}, \mathtt{x}, \pi) = 1 \wedge \neg(\exists \mathtt{w} : \mathcal{R}(\mathtt{x}, \mathtt{w}) = 1)$
then return 1; **else return** 0; **fi**

Fig. 14. Soundness experiment in Definition 5.

Definition 5. *A tuple of PPT algorithms* $\Pi = (\mathsf{K}_{\mathsf{bpk}}, \mathsf{PKV}, \mathsf{P}, \mathsf{V}, \mathsf{Sim})$ *is a zero knowledge (Sub-ZK) NIZK argument system in the BPK model if satisfies the following properties, for the experiments depicted in Figs. 12 to 14.*

Perfect completeness: *for all* λ, *PPT* $\mathcal{A}$, $\Pr[\mathsf{Complete}^{\mathsf{bpk}}_{\Pi,\mathcal{A}}(\lambda) = 1] = 1$.

Sub-ZK: $\forall$ *PPT subverters* $\mathcal{Z}$, *unbounded* $\mathcal{A}$, $\Pr[\mathsf{ZK}^{\mathsf{bpk}}_{\Pi,\mathcal{Z},\mathcal{A}}(\lambda) = 1] \approx_\lambda \frac{1}{2}$.

Computational soundness: $\forall$ *PPT* $\mathcal{A}$, $\Pr[\mathsf{Snd}^{\mathsf{bpk}}_{\Pi,\mathcal{A}}(\lambda) = 1] \approx_\lambda \frac{1}{2}$. Π *is* statistically sound *if this holds for all unbounded* $\mathcal{A}$.

E Omitted Proofs

E.1 Proof of Theorem 1

Proof. Let HF be a computationally-smooth κ-extractable auxiliary-string non-blackbox GL-SZKHF in the plain model for $\mathcal{L}_{\mathtt{lpar}}$. The execution of HF can be seen as a question hp from the verifier, who has access to the randomness hk, and an answer $\mathsf{pH} \leftarrow \mathsf{projhash}(\mathtt{lpar}; \mathtt{w}, \mathsf{hp}, \mathtt{x})$ by the prover. The prover's ability to provide an answer pH such that $\mathsf{pH} = \mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$ is seen as a sufficient evidence that $\mathtt{x} \in \mathcal{L}_{\mathtt{lpar}}$. The perfect completeness property of HF ensures that if $\mathtt{x} \in \mathcal{L}_{\mathtt{lpar}}$ then the prover will be able to output pH for any $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$. The computational-smoothness guarantees that if $\mathtt{x} \notin \mathcal{L}_{\mathtt{lpar}}$ then given only $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$, no PPT prover can distinguish $\mathsf{H} \leftarrow \mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$ from random for any but a negligible fraction of the hk's.

The idea of the proof is to run the $\mathsf{simhash}$ algorithm as a means of generating pH. To be able to do it so that we could still rely on the computational-smoothness, it is essential to hide from the extractor $\mathsf{Ext}_{\mathcal{Z}}$ the randomness used by the subverter $\mathcal{Z}$ when generating hp. This can be achieved by using the auxiliary string of the subverter as follows. Consider a subverter $\mathcal{Z}^*$ that, given a correctly generated projection key hpf as its auxiliary string, sets $\mathsf{hp} \leftarrow \mathsf{hpf}$ and outputs hp. Provided that the length of hpf is polynomial in the length of $\mathtt{x}$, $\mathcal{Z}^*$ is clearly a PPT machine. Thus, by the ZK property, there exists an extractor $\mathsf{Ext}_{\mathcal{Z}^*}$ that, given as input $\mathtt{x}$ and an auxiliary string hpf, outputs $\kappa(\mathsf{hk})$ without knowing the randomness of hpf. Using $\mathsf{Ext}_{\mathcal{Z}^*}$, we build a PPT adversary $\mathcal{B}$ that decides $\mathcal{L}_{\mathtt{lpar}}$. On input $\mathtt{x}_b$, $\mathcal{B}$ works as follows:

$\mathcal{B}(\mathtt{lpar}; \mathtt{x}_b)$ // $b = 0$ if $\mathtt{x} \in \mathcal{L}_{\mathtt{lpar}}$ and $b = 1$ if $\mathtt{x} \notin \mathcal{L}_{\mathtt{lpar}}$

$\mathsf{hk} \leftarrow \mathsf{hashkg}(\mathtt{lpar})$; $\mathsf{hpf} \leftarrow \mathsf{projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x}_b)$;
$r \leftarrow_{\$} \mathsf{RND}_\lambda(\mathcal{Z}^*)$; $(\mathsf{hpf}, \mathsf{st}_{\mathcal{Z}}) \leftarrow \mathcal{Z}^*(\mathtt{ltrap}; \mathtt{x}_b, \mathsf{aux} = \mathsf{hpf}; r)$;
$\kappa(\mathsf{hk}) \leftarrow \mathsf{Ext}_{\mathcal{Z}^*}(\mathtt{ltrap}; \mathtt{x}_b, \mathsf{aux}; r)$;
$\mathsf{H} \leftarrow \mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x}_b)$; $\mathsf{H}' \leftarrow \mathsf{simhash}(\mathtt{lpar}, \kappa(\mathsf{hk}); \mathsf{hpf}, \mathtt{x}_b)$;
if $\mathsf{H} = \mathsf{H}'$ **then** $b' \leftarrow 0$; **else** $b' \leftarrow 1$; **fi**
return b';

The soundness of $\mathcal{B}$ follows directly from the computational-smoothness of HF: if $\mathtt{x}_b \notin \mathcal{L}_{\mathtt{lpar}}$ and $\mathsf{simhash}$ is able to generate, with non-negligible probability, a hash value H' such that $\mathsf{H} = \mathsf{H}'$, then a PPT adversary $\mathcal{A}$ using $\mathsf{Ext}_{\mathcal{Z}^*}$ can

trivially break the computational-smoothness property. Thus, for any $\mathtt{x}_b \notin \mathcal{L}_{\mathtt{lpar}}$, $\mathcal{B}$ will output $b' = 1$ with high probability $1 - \varepsilon_{sm}$. Also, the ZK property of the HF guarantees the completeness of $\mathcal{B}$. Thus:

$$\Pr[b = b'] = (\Pr[b' = 0 | b = 0] + \Pr[b' = 1 | b = 1])/2 = \tfrac{1}{2} + \tfrac{1-\varepsilon_{sm}}{2} = 1 - \tfrac{\varepsilon_{sm}}{2} \ .$$

Thus, $\mathcal{B}$ has non-negligible advantage in deciding $\mathcal{L}_{\mathtt{lpar}}$. □

E.2 Proof of Theorem 2

We refer to Appendix A.2 for a brief overview of the algebraic group model (AGM).

Proof. **(1: SUBPAR-SZKHF-KE.)** The proof is inspired by that of the KWKE assumption in [ALSZ20]. However, the assumption itself is different. Moreover, we prove it in the standard AGM of [FKL18] instead of the HAK assumptions introduced in [Lip19]. This enables us to simplify the proof significantly.

Let $\mathcal{A}$ is a SUBPAR-SZKHF-KE adversary that, given public parameters p, and randomness $r \leftarrow\!\!\$\ \mathsf{RND}_\lambda(\mathcal{A})$ as input, outputs $\mathtt{lpar} = [\boldsymbol{\Gamma}(\mathtt{x})]_1$ and hpf, s.t. with probability $\epsilon_{\mathcal{A}}$, $\mathsf{verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 1$ and $\mathsf{verpar}(\mathtt{lpar}) = 1$. Denote $\boldsymbol{\Delta} := \tau\boldsymbol{\alpha}^\top \in \mathbb{Z}_p^{1\times n}$. Let $\mathsf{verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 1$, i.e., $[\boldsymbol{\Delta}]_2 \bullet [\boldsymbol{\Gamma}(\mathtt{x})]_1 = [\tau]_2 \bullet [\boldsymbol{\gamma}]_1 \in \mathbb{G}_T^{1\times k}$. Let $\mathsf{Ext}_{\mathcal{A}}^{\mathsf{agm}}$ be the extractor, existence of which is guaranteed by the AGM. Fig. 15 depicts the extractor $\mathsf{Ext}_{\mathcal{A}}$, who also emulates the oracle answers $[q_{\iota i}]_\iota$ for $i > 0$ to $\mathcal{A}$ in $\mathbb{G}_\iota$. $\mathsf{Ext}_{\mathcal{A}}^{\mathsf{agm}}$ extracts $\boldsymbol{N}_\iota$, such that

$$\left[\begin{smallmatrix}\mathrm{vect}(\boldsymbol{\Gamma})\\ \boldsymbol{\gamma}\end{smallmatrix}\right]_1 = \boldsymbol{N}_1 \left[\begin{smallmatrix}1\\ \boldsymbol{q}_1\end{smallmatrix}\right]_1 \in \mathbb{G}_1^{nk+k} \ , \qquad \left[\begin{smallmatrix}\tau\\ \mathrm{vect}(\boldsymbol{\Delta})\end{smallmatrix}\right]_2 = \boldsymbol{N}_2 \left[\begin{smallmatrix}1\\ \boldsymbol{q}_2\end{smallmatrix}\right]_2 \in \mathbb{G}_2^{n+1} \ .$$

Thus, e.g., $\tau = \sum_{t\geq 0}^{|\boldsymbol{q}_2|+1} N_{2,1,t}\boldsymbol{q}_{2t}$. Given $\boldsymbol{N}_1$, $\boldsymbol{N}_2$, $\boldsymbol{q}_1$, and $\boldsymbol{q}_2$, one can efficiently compute $\boldsymbol{\Gamma} \in \mathbb{Z}_p^{n\times k}$, $\boldsymbol{\gamma} \in \mathbb{Z}_p^{1\times k}$, $\tau \in \mathbb{Z}_p$ and $\boldsymbol{\Delta} \in \mathbb{Z}_p^{1\times n}$.

$\mathsf{Ext}_{\mathcal{A}}(\mathsf{p}; r)$

$\boldsymbol{q}_1 \leftarrow \emptyset; \boldsymbol{q}_2 \leftarrow \emptyset; \xi_1 \leftarrow 0; \xi_2 \leftarrow 0;$
$([\boldsymbol{\Gamma}]_1, \mathsf{hpf}) \leftarrow \mathcal{A}(\mathsf{p}; r);$
if $\mathsf{verpar}(\mathtt{lpar}) = 0 \vee \mathsf{verhp}(\mathtt{lpar}; \mathsf{hpf}, \mathtt{x}) = 0$ **then return** $\perp$; **fi** ;
$(\boldsymbol{N}_1, \boldsymbol{N}_2) \leftarrow \mathsf{Ext}_{\mathcal{A}}^{\mathsf{agm}}(\mathsf{p}; r)$; Abort if this fails;
Compute τ, $\boldsymbol{\Delta}$ from $\boldsymbol{N}_1$, $\boldsymbol{N}_2$, $\boldsymbol{q}_1$, $\boldsymbol{q}_2$;
return $\boldsymbol{\alpha} \leftarrow \tau^{-1}\boldsymbol{\Delta}$;

$\mathsf{O}(\iota)$

$\xi_\iota \leftarrow \xi_\iota + 1; q_{\iota\xi_\iota} \leftarrow\!\!\$\ \mathbb{Z}_p$; **return** $[q_{\iota\xi_\iota}]_\iota$;

Fig. 15. Extractors $\mathsf{Ext}_{\mathcal{A}}(\mathsf{p}; r)$ in the proof of Theorem 3

We will now show that $\mathsf{Ext}_{\mathcal{A}}$ satisfies the requirements of the extractor in Eq. (2). Assume that $\mathcal{A}(\mathsf{p}; r)$ was successful. We execute $\mathsf{Ext}_{\mathcal{A}}(\mathsf{p}; r)$ and obtain

either $\boldsymbol{\alpha}$ or $\perp$. From the fact that $[\boldsymbol{\Delta}]_2 \bullet [\boldsymbol{\Gamma}]_1 = [\tau]_2 \bullet [\boldsymbol{\gamma}]_1$, we get $\boldsymbol{\Delta\Gamma} = \tau\boldsymbol{\gamma} \in \mathbb{Z}_p^{1\times k}$. Since $\tau \neq 0$, $\boldsymbol{\gamma} = \tau^{-1}\boldsymbol{\Delta\Gamma} \in \mathbb{Z}_p^{1\times k}$. Clearly, $\boldsymbol{\alpha} := \tau^{-1}\boldsymbol{\Delta} \in \mathbb{Z}_p^n$ is a valid hk since $\boldsymbol{\alpha}^\top\boldsymbol{\Gamma} = \tau^{-1}\boldsymbol{\Delta\Gamma} = \boldsymbol{\gamma}$ and in particular $[\boldsymbol{\gamma}]_1 = \alpha^\top[\boldsymbol{\Gamma}]_1$.

(2: $\mathcal{D}_\mathsf{p}$-SZKHF-KE.) The proof is almost similar to the SUBPAR-SZKHF-KE proof with the difference that $\mathtt{lpar} = [\boldsymbol{\Gamma}]_1$ is honestly generated and so $\mathcal{A}$ is given $[\boldsymbol{\Gamma}]_1$ as additional input. □