Implicit Zero-Knowledge Arguments and Applications to the Malicious Setting

Fabrice Benhamouda, Geoffroy Couteau, David Pointcheval, and Hoeteck Wee

ENS, CNRS, INRIA, and PSL, Paris, France
`firstname.lastname@ens.fr`

Abstract. We introduce *implicit zero-knowledge* arguments (iZK) and simulation-sound variants thereof (SSiZK); these are lightweight alternatives to zero-knowledge arguments for enforcing semi-honest behavior. Our main technical contribution is a construction of efficient two-flow iZK and SSiZK protocols for a large class of languages under the (plain) DDH assumption in cyclic groups in the common reference string model. As an application of iZK, we improve upon the round-efficiency of existing protocols for securely computing inner product under the DDH assumption. This new protocol in turn provides privacy-preserving biometric authentication with lower latency.

Keywords. hash proof systems, zero-knowledge, malicious adversaries, two-party computation, inner product.

1 Introduction

Zero-Knowledge Arguments (ZK) enable a prover to prove the validity of a statement to a verifier without revealing anything else [13, 30]. In addition to being interesting in its own right, zero knowledge has found numerous applications in cryptography, most notably to simplify protocol design as in the setting of secure two-party computation [28, 29, 46], and as a tool for building cryptographic primitives with strong security guarantees such as encryption secure against chosen-ciphertext attacks [19, 41].

In this work, we focus on the use of zero-knowledge arguments as used in efficient two-party protocols for enforcing semi-honest behavior. We are particularly interested in round-efficient two-party protocols, as network latency and round-trip times can be a major efficiency bottleneck, for instance, when a user wants to securely compute on data that is outsourced to the cloud. In addition, we want to rely on standard and widely-deployed cryptographic assumptions. Here, a standard interactive zero-knowledge argument based on the DDH assumption would require at least three flows; moreover, this overhead in round complexity is incurred each time we want to enforce semi-honest behavior via zero knowledge. To avoid this overhead, we could turn to non-interactive zero-knowledge proofs (NIZK). However, efficient NIZK would require either the use of pairings [32] and thus stronger assumptions and additional efficiency overhead, or the use of random oracles [6, 23].

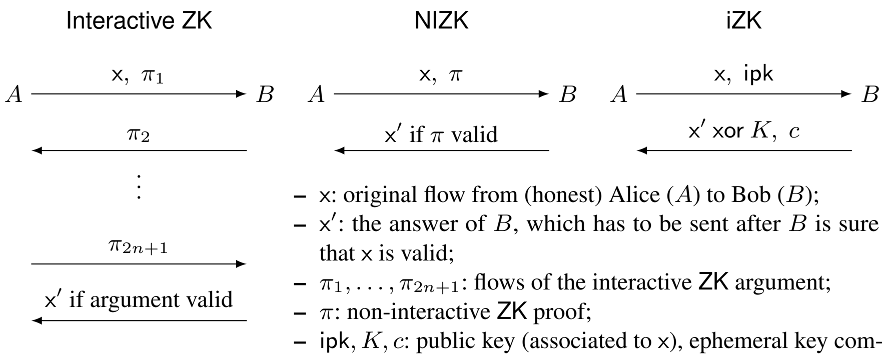
We would like to point out that, contrary to some common belief, there is no straightforward way to reduce the number of rounds of zero-knowledge proofs "à la Schnorr" [42] by performing the first steps (commitment and challenges) in a preprocessing phase, so that each proof only takes one flow subsequently. Indeed, as noticed by Bernhard-Pereira-Warinsky in [9], the statement of the proof has to be chosen before seeing the challenges, unless the proof becomes unsound.

On the Importance of Round-Efficiency. In addition to being an interesting theoretical problem, improving the round efficiency is also very important in practice. If we consider a protocol between a client in Europe, and a cloud provider in the US, for example, we expect a latency of at least 100ms (and even worse if the client is connected with 3g or via satellite, which may induce a latency of up to 1s [14]). Concretely, using Curve25519 elliptic curve of Bernstein [10] (for 128 bits of security, and 256-bit group elements) with a 10Mbps Internet link and 100ms latency, 100ms corresponds to sending 1 flow, or 40,000 group elements, or computing 1,000 exponentiations at 2GHz on one core of current AMD64 microprocessor[1], hence 4,000 exponentiations on a 4-core microprocessor[2]. As a final remark on latency, while speed of networks keeps increasing as technology improves, latency between two (far away) places on earth is strongly limited by the speed of light: there is no hope to get a latency less than 28ms between London and San Francisco, for example.

Our Contributions. In this work, we introduce *implicit Zero-Knowledge Arguments* or iZK and simulation-sound variants thereof or SSiZK, lightweight alternatives to (simulation-sound) zero-knowledge arguments for enforcing semi-honest behavior in two-party protocols. Then, we construct efficient two-flow iZK and SSiZK protocols for a large class of languages under the (plain) DDH assumption in cyclic groups without random oracles; this is the main technical contribution of our work. Our SSiZK construction from iZK is very efficient and incurs only a small additive overhead. Finally, we present several applications of iZK to the design of efficient secure two-party computation, where iZK can be used in place of interactive zero-knowledge arguments to obtain more round-efficient protocols.

While our iZK protocols require an additional flow compared to NIZK, we note that eliminating the use of pairings and random oracles offers both theoretical and practical benefits. From a theoretical stand-point, the DDH assumption in cyclic groups is a weaker assumption than the DDH-like assumptions used in Groth-Sahai pairing-based NIZK [32], and we also avoid the theoretical pitfalls associated with instantiating the random oracle methodology [5, 16]. From a practical stand-point, we can instantiate our DDH-based protocols over a larger class of groups. Concrete examples include Bernstein's Curve25519 [10] which admit very efficient group exponentiations, but do not support an efficient pairing and are less likely to be susceptible to recent breakthroughs in discrete log attacks [4, 31]. By using more efficient groups and avoiding the use of pairing operations, we also gain notable improvements in computational efficiency over Groth-Sahai proofs. Moreover, additional efficiency improvements come from the structure of iZK which makes them *efficiently batchable*. Conversely, Groth-Sahai NIZK cannot be efficiently batched and do not admit efficient SS-NIZK (for non-linear equations).

[1] According to [20], an exponentiation takes about 200,000 cycles.
[2] Assuming exponentiations can be made in parallel, which is the case for our iZKs.


Interactive ZK
NIZK
iZK
A
B
$\mathsf{x},\ \pi_1$
π_2
π_{2n+1}
x' if argument valid
$\mathsf{x},\ \pi$
x' if π valid
$\mathsf{x},\ \mathsf{ipk}$
x' xor $K,\ c$

- x: original flow from (honest) Alice (A) to Bob (B);
- x': the answer of B, which has to be sent after B is sure that x is valid;
- $\pi_1, \ldots, \pi_{2n+1}$: flows of the interactive ZK argument;
- π: non-interactive ZK proof;
- ipk, K, c: public key (associated to x), ephemeral key computed by B, key encapsulation (which can be decapsulated by A if she generated honestly ipk, using a witness that x was valid), respectively.

Fig. 1: Enforcing semi-honest behavior of Alice (A)

New Notion: Implicit Zero-Knowledge Arguments. iZK is a two-party protocol executed between a prover and a verifier, at the end of which both parties should output an ephemeral key. The idea is that the key will be used to encrypt subsequent messages and to protect the privacy of a verifier against a cheating prover. Completeness states that if both parties start with a statement in the language, then both parties output the same key K. Soundness states that if the statement is outside the language, then the verifier's ephemeral output key is hidden from the cheating prover. Note that the verifier may not learn whether his key is the same as the prover's and would not be able to detect whether the prover is cheating, hence the soundness guarantee is *implicit*. This is in contrast to a standard ZK argument, where the verifier would "explicitly" abort when interacting with a cheating prover. Finally, zero-knowledge stipulates that for statements in the language, we can efficiently simulate (without the witness) the joint distribution of the transcript between an honest prover and a malicious verifier, together with the honest prover's ephemeral output key K. Including K in the output of the simulator ensures that the malicious verifier does not gain additional knowledge about the witness when honest prover uses K in subsequent interaction, as will be the case when iZK is used as part of a bigger protocol.

More precisely, iZK are key encapsulation mechanisms in which the public key ipk is associated with a word x and a language $\mathsf{i}\mathscr{L}$. In our case, x is the flow[3] and $\mathsf{i}\mathscr{L}$ the language of valid flows. If x is in $\mathsf{i}\mathscr{L}$, knowing a witness proving so (namely, random coins used to generate the flow) enables anyone to generate ipk together with a secret key isk, using a key generation algorithm iKG. But, if x is not in $\mathsf{i}\mathscr{L}$, there is no polynomial-time way to generate a public key ipk for which it is possible to decrypt the associated ciphertexts (*soundness*).

[3] In our formalization, actually, it is the flow together all the previous flows. But we just say it is the flow to simplify explanations.

To ensure semi-honest behavior, as depicted in Figure 1, each time a player sends a flow x, he also sends a public key ipk generated by iKG and keeps the associated secret key isk. To answer back, the other user generates a key encapsulation c for ipk and x, of a random ephemeral key K. He can then use K to encrypt (using symmetric encryption or pseudo-random generators and one-time pad) all the subsequent flows he sends to the first player. For this transformation to be secure, we also need to be sure that c (and the ability to decapsulate K for any ipk) leaks no information about random coins used to generate the flow (or, more generally, the witness of x). This is ensured by the *zero-knowledge* property, which states there must exist a trapdoor (for some common reference string) enabling to generate a public key ipk and a trapdoor key itk (using a trapdoor key algorithm iTKG), so that ipk looks like a classical public key and itk allows to decapsulate any ciphertext for ipk.

Overview of our iZK and SSiZK Constructions. We proceed to provide an overview of our two-flow iZK protocols; this is the main technical contribution of our work. Our main tool is Hash Proof Systems or Smooth Projective Hash Functions (SPHFs) [18]. We observe that SPHFs are essentially "honest-verifier" iZK; our main technical challenge is to boost this weak honest-verifier into full-fledged zero knowledge, without using pairings or random oracles.

Informally speaking, a smooth projective hash function on a language $\mathscr{L}$ is a sort of hash function whose evaluation on a word $C \in \mathscr{L}$ can be computed in two ways, either by using a *hashing key* hk (which can be seen as a private key) or by using the associated *projection key* hp (which can be seen as a public key). On the other hand, when $C \notin \mathscr{L}$, the hash of C cannot be computed from hp; actually, when $C \notin \mathscr{L}$, the hash of C computed with hk is statistically indistinguishable from a random value from the point of view of any individual knowing the projection key hp only. Hence, an SPHF on $\mathscr{L}$ is given by a pair $(\mathsf{Hash}, \mathsf{ProjHash})$ with the requirements that, when there is a witness w ensuring that $C \in \mathscr{L}$, $\mathsf{Hash}(\mathsf{hk}, \mathscr{L}, C) = \mathsf{ProjHash}(\mathsf{hp}, \mathscr{L}, C, w)$, while when there is no such witness (i.e. $C \notin \mathscr{L}$), the smoothness property states that $H = \mathsf{Hash}(\mathsf{hk}, \mathscr{L}, C)$ is random and independent of hp. In this paper, as in [26], we consider a weak form of SPHFs, where the projection key hp can depend on C.

Concretely, if we have an SPHF for some language $\mathscr{L}$, we can set the public key ipk to be empty ($\perp$), the secret key isk to be the witness w, the ciphertext c to be the projection key hp, and the encapsulated ephemeral key K would be the hash value. (Similar connections between SPHF and zero knowledge were made in [1, 12, 25, 26].) The resulting iZK would be correct and sound, the soundness coming from the smoothness of the SPHF: if the word C is not in $\mathscr{L}$, even given the ciphertext $c = \mathsf{hp}$, the hash value K looks random. However, it would not necessarily be zero-knowledge for two reasons: not only, a malicious verifier could generate a malformed projection key, for which the projected hash value of a word depends on the witness, but also there seems to be no trapdoor enabling to compute the hash value K from only $c = \mathsf{hp}$.

These two issues could be solved using either Trapdoor SPHF [7] or NIZK of knowledge of hk. But both methods require pairings or random oracle, if instantiated on cyclic or bilinear groups. Instead we construct it as follows:

First, suppose that a projection key is well-formed (i.e., there exists a corresponding hashing key). Then, there exists an *unbounded* zero-knowledge simulator that "extracts" a corresponding hashing key and computes the hash value. To boost this into full-fledged zero knowledge with an efficient simulator, we rely on the "OR trick" from [22]. We add a random 4-tuple (g', h', u', e') to the CRS, and build an SPHF for the augmented language $C \in \mathscr{L}$ or (g', h', u', e') is a DDH tuple. In the normal setup, (g', h', u', e') is not a DDH tuple with overwhelming probability, so the soundness property is preserved. In the trapdoor setup, $(g', h', u', e') := (g', h', g'^r, h'^r)$ is a random DDH tuple, and the zero-knowledge simulator uses the witness r to compute the hash value.

Second, to ensure that the projection key is well-formed, we use a second SPHF. The idea for building the second SPHF is as follows: in most SPHF schemes, proving that a projected key hp is valid corresponds to proving that it lies in the column span of some matrix Γ (where all of the linear algebra is carried out in the exponent). Now pick a random vector tk: if hp lies in the span of Γ, then $\mathsf{hp}^\intercal \mathsf{tk}$ is completely determined given $\Gamma^\intercal \mathsf{tk}$; otherwise, it is completely random. The former yields the projective property and the latter yields smoothness, for the SPHF with hashing key hk and projection key $\mathsf{tp} = \Gamma^\intercal \mathsf{tk}$. Since the second SPHF is built using the transpose $\Gamma^\intercal$ of the original matrix Γ (defining the language $\mathscr{L}$), we refer to it as a "transpose SPHF". As it turns out, the second fix could ruin soundness of the ensuing iZK protocol: a cheating prover could pick a malformed $\Gamma^\intercal \mathsf{tk}$, and then the hash value $\mathsf{hp}^\intercal \mathsf{tk}$ computed by the verifier could leak additional information about his witness hk for hp, thereby ruining smoothness. To protect against the leakage, we would inject additional randomness into hk so that smoothness holds even in the presence of leakage from the hash value $\mathsf{hp}^\intercal \mathsf{tk}$. This idea is inspired by the 2-universality technique introduced in a very different context of chosen-ciphertext security [18].

Finally, to get simulation-soundness (i.e., soundness even if the adversary can see fake or simulated proofs), we rely on an additional "OR trick" (mixed up with an idea of Malkin et al. [40]): we build an SPHF for the augmented language $C \in \mathscr{L}$, or (g', h', u', e') is a DDH tuple (as before), or $(g', h', \mathcal{W}_1(C), \mathcal{W}_2(C))$ is not a DDH tuple (with $\mathcal{W}_k$ a Waters function [45], $\mathcal{W}_k(m) = v_{k,0} \prod_{i=1}^{|m|} v_{k,i}^{m_i}$, when $m = m_1 \| \ldots \| m_{|m|}$ is a bitstring, the $v_{k,0}, \ldots, v_{k,|m|}$ are random group elements, and C is seen as a bitstring, for $k = 1, 2$). In the security proof, with non-negligible probability, $(g'', h'', \mathcal{W}_1(C), \mathcal{W}_2(C))$ is a non-DDH tuple for simulated proofs, and a DDH tuple for the soundness challenge, which proves simulation-soundness.

Organization. First, we formally introduce the notion of *implicit zero-knowledge proofs* (iZK) in Section 2. Second, in Section 3, we discuss some difficulties related to the construction of iZK from SPHF and provide an intuition of our method to overcome these difficulties. Next, we show how to construct iZK and SSiZK from SPHF over cyclic groups for any language handled by the generic framework [7], which encompasses most, if not all, known SPHFs over cyclic groups. This is the main technical part of the paper. Third, in Section 4, we indeed show a concrete application of our iZK constructions: the most efficient 3-round two-party protocol computing inner product in the UC framework with static corruption so far. We analyze our construction and provide a detailed comparison with the Groth-Sahai methodology [32] and the approach based on zero-knowledge proofs "à la Schnorr" [42]. In addition, as proof of concept, we show in the full version [8] that iZK can be used instead of ZK arguments to generically convert any protocol secure in the semi-honest model into a protocol secure in the malicious model. This conversion follows the generic transformation of Goldreich, Micali and Wigderson (GMW) in their seminal papers [28,29]. While applying directly the original transformation with Schnorr-like ZK protocols blows up the number of rounds by a multiplicative factor of at least three (even in the common reference string model), our conversion only adds a small constant number of rounds. Eventually, in the full version [8], we extend our construction of iZK from SPHF to handle larger classes of languages described by computational structures such as circuits or branching programs.

Additional Related Work. Using the "OR trick" with SPHF is reminiscent of [2]. However, the methods used in our paper are very different from the one in [2], as we do not use pairings, but consider weaker form of SPHF on the other hand.

A recent line of work has focused on the cut-and-choose approach for transforming security from semi-honest to malicious models [34,35,37–39,43,44] as an alternative to the use of zero-knowledge arguments. Indeed, substantial progress has been made towards practical protocols via this approach, as applied to Yao's garbled circuits. However, the state-of-the-art still incurs a large computation and communication multiplicative overhead that is equal to the security parameter. We note that Yao's garbled circuits do not efficiently generalize to arithmetic computations, and that our approach would yield better concrete efficiency for natural functions F that admit compact representations by arithmetic branching programs. In particular, Yao's garbled circuits cannot take advantage of the structure in languages handled by the Groth-Sahai methodology [32], and namely the ones defined by multi-exponentiations: even in the latter case, Groth-Sahai technique requires pairings, while we will be able to avoid them.

The idea of using implicit proofs (without the zero-knowledge requirement) as a lightweight alternative to zero-knowledge proofs also appeared in an earlier work of Aiello, Ishai and Reingold [3]. They realize implicit proofs using conditional disclosure of secrets [27]. The latter, together with witness encryption [24] and SPHFs, only provide a weak "honest-verifier zero-knowledge" guarantee.

Recently, Jarecki introduced the concept of conditional key encapsulation mechanism [36], which is related to iZK as it adds a "zero-knowledge flavor" to SPHFs by allowing witness extraction. The construction is a combination of SPHF and zero-knowledge proofs "à la Schnorr". Contrary to iZK, it does not aim at reducing the interactivity of the resulting protocol, but ensures its covertness.

Witness encryption was introduced by Garg et al. in [24]. It enables to encrypt a message M for a word C and a language $\mathscr{L}$ into a ciphertext c, so that any user knowing a witness w that $C \in \mathscr{L}$ can decrypt c. Similarly to SPHFs, witness encryption also only has this "honest-verifier zero-knowledge" flavor: it does not enable to decrypt ciphertext for words $C \notin \mathscr{L}$, with a trapdoor. That is why, as SPHF, witness encryption cannot be used to construct directly iZK.

2 Definition of Implicit Zero-Knowledge Arguments

2.1 Notations

Since we will now be more formal, let us present the notations that we will use. Let $\{0,1\}^*$ be the set of bitstrings. We denote by PPT a probabilistic polynomial time algorithm. We write $y \leftarrow A(x)$ for 'y is the output of the algorithm A on the input x', while $y \overset{\$}{\leftarrow} A(x)$ means that A will additionally use random coins. Similarly, $X \overset{\$}{\leftarrow} \mathcal{X}$ indicates that X has been chosen uniformly at random in the (finite) set $\mathcal{X}$. We sometimes write st the state of the adversary. We define, for a distinguisher A and two distributions $\mathcal{D}_0, \mathcal{D}_1$, the advantage of A (i.e., its ability to distinguish those distributions) by $\mathsf{Adv}^{\mathcal{D}_0,\mathcal{D}_1}(A) = \Pr_{x \in \mathcal{D}_0}[A(x) = 1] - \Pr_{x \in \mathcal{D}_1}[A(x) = 1]$. The qualities of adversaries will be measured by their successes and advantages in certain experiments $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A}}$ or $\mathsf{Exp}^{\mathsf{sec}-b}_{\mathcal{A}}$: $\mathsf{Succ}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A}}(1^{\mathfrak{K}}) = 1]$ and $\mathsf{Adv}^{\mathsf{sec}}(\mathcal{A}, \mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{sec}-1}_{\mathcal{A}}(1^{\mathfrak{K}}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sec}-0}_{\mathcal{A}}(1^{\mathfrak{K}}) = 1]$ respectively, where $\mathfrak{K}$ is the security parameter, and probabilities are over the random coins of the challenger and of the adversary.

2.2 Definition

Let $(\mathsf{i}\mathscr{L}_{\mathsf{crs}})_{\mathsf{crs}}$ be a family of NP languages, indexed by a common reference string crs, and defined by a witness relation $\mathsf{i}\mathcal{R}_{\mathsf{crs}}$, namely $\mathsf{i}\mathscr{L} = \{\mathsf{x} \in \mathsf{i}\mathcal{X}_{\mathsf{crs}} \mid \exists \mathsf{iw},\, \mathsf{i}\mathcal{R}_{\mathsf{crs}}(\mathsf{x}, \mathsf{iw}) = 1\}$, where $(\mathsf{i}\mathcal{X}_{\mathsf{crs}})_{\mathsf{crs}}$ is a family of sets. crs is generated by some polynomial-time algorithm $\mathsf{Setup}_{\mathsf{crs}}$ taking as input the unary representation of the security parameter $\mathfrak{K}$. We suppose that membership to $\mathcal{X}_{\mathsf{crs}}$ and $\mathsf{i}\mathcal{R}_{\mathsf{crs}}$ can be evaluated in polynomial time (in $\mathfrak{K}$). For the sake of simplicity, crs is often implicit.

To achieve stronger properties (namely simulation-soundness in Section 3.4), we sometimes also assume that $\mathsf{Setup}_{\mathsf{crs}}$ can also output some additional information or trapdoor $\mathcal{T}_{\mathsf{crs}}$. This trapdoor should enable to check, in polynomial time, whether a given word x is in $\mathsf{i}\mathscr{L}$ or not. It is only used in security proofs, and is never used by the iZK algorithms.

An iZK is defined by the following polynomial-time algorithms:

- $\mathsf{icrs} \overset{\$}{\leftarrow} \mathsf{iSetup}(\mathsf{crs})$ generates the (normal) common reference string (CRS) icrs (which implicitly contains crs). The resulting CRS provides statistical soundness;
- $(\mathsf{icrs}, \mathsf{i}\mathcal{T}) \overset{\$}{\leftarrow} \mathsf{iTSetup}(\mathsf{crs})$[4] generates the (trapdoor) common reference string icrs together with a trapdoor $\mathsf{i}\mathcal{T}$. The resulting CRS provides statistical zero-knowledge;
- $(\mathsf{ipk}, \mathsf{isk}) \overset{\$}{\leftarrow} \mathsf{iKG}^{\ell}(\mathsf{icrs}, \mathsf{x}, \mathsf{iw})$ generates a public/secret key pair, associated to a word $\mathsf{x} \in \mathsf{i}\mathscr{L}$ and a label $\ell \in \{0,1\}^*$, with witness iw;
- $(\mathsf{ipk}, \mathsf{itk}) \overset{\$}{\leftarrow} \mathsf{iTKG}^{\ell}(\mathsf{icrs}, \mathsf{i}\mathcal{T}, \mathsf{x})$ generates a public/trapdoor key pair, associated to a word $\mathsf{x} \in \mathcal{X}$ and a label $\ell \in \{0,1\}^*$;
- $(c, K) \overset{\$}{\leftarrow} \mathsf{iEnc}^{\ell}(\mathsf{icrs}, \mathsf{ipk}, \mathsf{x})$ outputs a ciphertext c of a value K (an ephemeral key), for the public key ipk, the word x, and the label $\ell \in \{0,1\}^*$;
- $K \leftarrow \mathsf{iDec}^{\ell}(\mathsf{icrs}, \mathsf{isk}, c)$ decrypts the ciphertext c for the label $\ell \in \{0,1\}^*$, and outputs the ephemeral key K;
- $K \leftarrow \mathsf{iTDec}^{\ell}(\mathsf{icrs}, \mathsf{itk}, c)$ decrypts the ciphertext c for the label $\ell \in \{0,1\}^*$, and outputs the ephemeral key K.

[4] When the CRS is word-dependent, i.e., when the trapdoor $\mathsf{i}\mathcal{T}$ does only work for one word x^* previously chosen, there is a second argument: $(\mathsf{icrs}, \mathsf{i}\mathcal{T}) \overset{\$}{\leftarrow} \mathsf{iTSetup}(\mathsf{crs}, \mathsf{x}^*)$. Security notions are then slightly different. See details in the full version [8].

The three last algorithms can be seen as key encapsulation and decapsulation algorithms. Labels ℓ are only used for SSiZK and are often omitted. The CRS icrs is often omitted, for the sake of simplicity.

Normally, the algorithms iKG and iDec are used by the user who wants to (implicitly) prove that some word x is in $\mathsf{i}\mathscr{L}$ (and we often call this user the prover), while the algorithm iEnc is used by the user who wants to (implicitly) verify this (and we often call this user the verifier), as shown in Figs. 1 and 3. The algorithms iTKG and iTDec are usually only used in proofs, to generate simulated or fake implicit proofs (for the zero-knowledge property).

2.3 Security Requirements

An iZK satisfies the four following properties (for any $(\mathsf{crs}, \mathcal{T}_\mathsf{crs}) \overset{\$}{\leftarrow} \mathsf{Setup}_\mathsf{crs}(1^\mathfrak{K})$):

- **Correctness.** The encryption is the reverse operation of the decryption, with or without a trapdoor: for any $\mathsf{icrs} \overset{\$}{\leftarrow} \mathsf{iSetup}(\mathsf{crs})$ or with a trapdoor, for any $(\mathsf{icrs}, \mathsf{i}\mathcal{T}) \overset{\$}{\leftarrow} \mathsf{iTSetup}(\mathsf{crs})$, and for any $\mathsf{x} \in \mathcal{X}$ and any $\ell \in \{0,1\}^*$,
  - if $\mathsf{x} \in \mathsf{i}\mathscr{L}$ with witness iw, $(\mathsf{ipk}, \mathsf{isk}) \overset{\$}{\leftarrow} \mathsf{iKG}^\ell(\mathsf{icrs}, \mathsf{x}, \mathsf{iw})$, and $(c, K) \overset{\$}{\leftarrow} \mathsf{iEnc}^\ell(\mathsf{ipk}, \mathsf{x})$, then we have $K = \mathsf{iDec}^\ell(\mathsf{isk}, c)$;
  - if $(\mathsf{ipk}, \mathsf{itk}) \overset{\$}{\leftarrow} \mathsf{iTKG}^\ell(\mathsf{i}\mathcal{T}, \mathsf{x})$ and $(c, K) \overset{\$}{\leftarrow} \mathsf{iEnc}^\ell(\mathsf{ipk}, \mathsf{x})$, then we have $K = \mathsf{iTDec}^\ell(\mathsf{itk}, c)$.
- **Setup Indistinguishability.** A polynomial-time adversary cannot distinguish a normal CRS generated by iSetup from a trapdoor CRS generated by $\mathsf{iTSetup}$. More formally, no PPT can distinguish, with non-negligible advantage, the two distributions:

$$\{\mathsf{icrs} \mid \mathsf{icrs} \overset{\$}{\leftarrow} \mathsf{iSetup}(\mathsf{crs})\} \qquad \{\mathsf{icrs} \mid (\mathsf{icrs}, \mathsf{i}\mathcal{T}) \overset{\$}{\leftarrow} \mathsf{iTSetup}(\mathsf{crs})\}.$$

- **Soundness.** When the CRS is generated as $\mathsf{icrs} \overset{\$}{\leftarrow} \mathsf{iSetup}(\mathsf{crs})$, and when $\mathsf{x} \notin \mathscr{L}$, the distribution of K is statistically indistinguishable from the uniform distribution, even given c. More formally, if Π is the set of all the possible values of K, for any bitstring ipk, for any word $\mathsf{x} \notin \mathsf{i}\mathscr{L}$, for any label $\ell \in \{0,1\}^*$, the two distributions:

$$\{(c, K) \mid (c, K) \overset{\$}{\leftarrow} \mathsf{iEnc}^\ell(\mathsf{ipk}, \mathsf{x})\} \quad \{(c, K') \mid (c, K) \overset{\$}{\leftarrow} \mathsf{iEnc}^\ell(\mathsf{ipk}, \mathsf{x}); K' \overset{\$}{\leftarrow} \Pi\}$$

  are statistically indistinguishable (iEnc may output $(\perp, K)$ when the public key ipk is not well formed).
- **Zero-Knowledge.** For any label $\ell \in \{0,1\}^*$, when the CRS is generated using $(\mathsf{icrs}, \mathsf{i}\mathcal{T}) \overset{\$}{\leftarrow} \mathsf{iTSetup}^\ell(\mathsf{crs})$, for any message $\mathsf{x}^* \in \mathsf{i}\mathscr{L}$ with the witness iw^*, the public key ipk and the decapsulated key K corresponding to a ciphertext c chosen by the adversary, either using isk or the trapdoor itk, should be indistinguishable, even given the trapdoor $\mathsf{i}\mathcal{T}$. More formally, we consider the experiment $\mathsf{Exp}^{\mathtt{iZK\text{-}zk\text{-}}b}$ in Figure 2. The iZK is (statistically) zero-knowledge if the advantage of any adversary $\mathcal{A}$ (not necessarily polynomial-time) for this experiment is negligible.

```
$\mathsf{Exp}^{\texttt{iZK-zk-}b}(\mathcal{A}, \mathsf{crs}, \mathfrak{K})$
    $(\mathsf{icrs}, \mathsf{i}\mathcal{T}) \xleftarrow{\$} \mathsf{iTSetup}(\mathsf{crs})$
    $(\ell, \mathsf{x}^*, \mathsf{iw}, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(\mathsf{icrs}, \mathsf{i}\mathcal{T})$
    **if** $\mathsf{i}\mathcal{R}(\mathsf{x}^*, \mathsf{iw}) = 0$ **then return** random bit
    **if** $b = 0$ **then** $(\mathsf{ipk}, \mathsf{isk}) \xleftarrow{\$} \mathsf{iKG}^{\ell}(\mathsf{icrs}, \mathsf{x}^*, \mathsf{iw}^*)$
    **else** $(\mathsf{ipk}, \mathsf{itk}) \xleftarrow{\$} \mathsf{iTKG}^{\ell}(\mathsf{i}\mathcal{T}, \mathsf{x}^*)$
    $(c, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(\mathsf{st}, \mathsf{icrs}, \mathsf{i}\mathcal{T}, \mathsf{ipk})$
    **if** $b = 0$ **then** $K \leftarrow \mathsf{iDec}^{\ell}(\mathsf{isk}, c)$
    **else** $K \leftarrow \mathsf{iTDec}^{\ell}(\mathsf{itk}, c)$
    **return** $\mathcal{A}(\mathsf{st}, K)$
```

```
$\mathsf{Exp}^{\texttt{iZK-ss-}b}(\mathcal{A}, \mathsf{crs}, \mathfrak{K})$
    $(\mathsf{icrs}, \mathsf{i}\mathcal{T}) \xleftarrow{\$} \mathsf{iTSetup}(\mathsf{crs})$
    $(\ell^*, \mathsf{x}^*, \mathsf{ipk}, \mathsf{st}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\mathsf{icrs})$
    $(c, K) \xleftarrow{\$} \mathsf{iEnc}^{\ell}(\mathsf{ipk}, \mathsf{x}^*)$
    **if** $b = 0$ **then** $K' \leftarrow K$
    **else** $K' \xleftarrow{\$} \Pi$
    $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\mathsf{st}, c, K')$
    **if** $\exists \mathsf{itk},\ (\ell^*, \mathsf{x}^*, \mathsf{ipk}, \mathsf{itk}) \in L \cup L'$ **then**
        **return** random bit
    **if** $\mathsf{x}^* \in \mathsf{i}\mathscr{L}$ **then return** random bit
    **return** $b'$
```

Fig. 2: Experiments $\mathsf{Exp}^{\texttt{iZK-zk-}b}$ for zero-knowledge of iZK, and $\mathsf{Exp}^{\texttt{iZK-ss-}b}$ for simulation-soundness of SSiZK

$$\begin{array}{rcl}
\text{Prover } \mathcal{P} & & \text{Verifier } \mathcal{V} \\
(\mathsf{ipk}, \mathsf{isk}) \xleftarrow{\$} \mathsf{iKG}(\mathsf{icrs}, \mathsf{x}, \mathsf{iw}) & \xrightarrow{\quad \mathsf{x},\ \mathsf{ipk} \quad} & \\
 & \xleftarrow{\quad c \quad} & (c, K) \xleftarrow{\$} \mathsf{iEnc}(\mathsf{ipk}, \mathsf{x}) \\
K' \leftarrow \mathsf{iDec}(\mathsf{isk}, c) & \xrightarrow{\quad K' \quad} & \text{accept if } K' = K
\end{array}$$

Fig. 3: Three-round zero-knowledge from iZK for a word $\mathsf{x} \in \mathsf{i}\mathscr{L}$ and a witness iw

We defined our security notion with a "composable" security flavor, as Groth and Sahai in [32]: soundness and zero-knowledge are statistical properties, the only computational property is the setup indistinguishability property. This is slightly stronger than what is needed, but is satisfied by our constructions and often easier to use.

We also consider stronger iZK, called simulation-sound iZK or SSiZK, which satisfies the following additional property:

- **Simulation Soundness.** The soundness holds (computationally) even when the adversary can see simulated public keys and decryption with these keys. More formally, we consider the experiment $\mathsf{Exp}^{\texttt{iZK-ss-}b}$ in Figure 2, where the oracle $\mathcal{O}$, and the lists L and L' are defined as follows:
  - on input (ℓ, x), $\mathcal{O}$ generates $(\mathsf{ipk}, \mathsf{itk}) \xleftarrow{\$} \mathsf{iTKG}(\mathsf{icrs}, \mathsf{i}\mathcal{T}, \mathsf{x})$, stores $(\ell, \mathsf{x}, \mathsf{ipk}, \mathsf{itk})$ in a list L, and outputs ipk;
  - on input (ipk, c), $\mathcal{O}$ retrieves the record $(\ell, \mathsf{x}, \mathsf{ipk}, \mathsf{itk})$ from L (and aborts if no such record exists), removes it from L, and adds it to L', computes $K \leftarrow \mathsf{iTDec}^{\ell}(\mathsf{icrs}, \mathsf{itk}, c)$, and outputs K.

  The iZK is (statistically) simulation-sound if the advantage of any adversary $\mathcal{A}$ (not necessarily polynomial-time) for this experiment is negligible.

Remark 1. An iZK for some language $\mathsf{i}\mathscr{L}$ directly leads to a 3-round zero-knowledge arguments for $\mathsf{i}\mathscr{L}$. The construction is depicted in Fig. 3 and the proof is provided in the full version [8]. If the iZK is additionally simulation-sound, the resulting zero-knowledge argument is also simulation-sound.

Remark 2. For the sake of completeness, in the full version [8], we show how to construct iZK from either NIZK or Trapdoor SPHFs. In the latter case, the resulting iZK is not statistically sound and zero-knowledge but only computationally sound and zero-knowledge. In both cases, using currently known constructions over cyclic groups, strong assumptions such as the random oracle model or pairings are needed.

3 Construction of Implicit Zero-Knowledge Arguments

Let us first recall the generic framework of SPHFs [7] for the particular case of cyclic groups, and when the projection key hp can depend on the word C, as it is at the core of our construction of iZK. Second, we explain in more details the limitations of SPHFs and the fact they cannot directly be used to construct iZK (even with a concrete attack). Third, we show how to overcome these limitations to build iZK and SSiZK.

3.1 Review of the Generic Framework of SPHFs over Cyclic Groups

Languages. Let $\mathbb{G}$ be a cyclic group of prime order p and $\mathbb{Z}_p$ the field of integers modulo p. If we look at $\mathbb{G}$ and $\mathbb{Z}_p$ as the same ring $(\mathbb{G}, +, \bullet)$, where internal operations are on the scalars, many interesting languages can be represented as subspaces of the vector space $\mathbb{G}^n$, for some n. Here are some examples.

Example 3 (DDH or ElGamal ciphertexts of 0*).* Let g and h be two generators of $\mathbb{G}$. The language of DDH tuples in basis (g, h) is

$$\mathscr{L} = \{(u, e) \in \mathbb{G}^2 \mid \exists r \in \mathbb{Z}_p,\ u = g^r \text{ and } e = h^r\} \subseteq \mathbb{G}^2,$$

where r is the witness. It can be seen as the subspace of $\mathbb{G}^2$ generated by (g, h). We remark that this language can also be seen as the language of (additive) ElGamal ciphertexts of 0 for the public key $\mathsf{pk} = (g, h)$. □

Example 4 (ElGamal ciphertexts of a bit). Let us consider the language of ElGamal ciphertexts of 0 or 1, under the public key $\mathsf{pk} = (g, h)$:

$$\mathscr{L} := \{(u, e) \in \mathbb{G}^2 \mid \exists r \in \mathbb{Z}_p, \exists b \in \{0, 1\},\ u = g^r \text{ and } e = h^r g^b\}.$$

Here $C = (u, e)$ cannot directly be seen as an element of some vector space. However, a word $C = (u, e) \in \mathbb{G}^2$ is in $\mathscr{L}$ if and only there exists $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \lambda_3) \in \mathbb{Z}_p^3$ such that:

$$\begin{aligned} u &= g^{\lambda_1}\ (= \lambda_1 \bullet g) & e &= h^{\lambda_1} g^{\lambda_2}\ (= \lambda_1 \bullet h + \lambda_2 \bullet g) \\ 1 &= u^{\lambda_2} g^{\lambda_3}\ (= \lambda_2 \bullet u + \lambda_3 \bullet g) & 1 &= (e/g)^{\lambda_2} h^{\lambda_3}\ (= \lambda_2 \bullet (e - g) + \lambda_3 \bullet h), \end{aligned}$$

because, if we write $C = (u, e) = (g^r, h^r g^b)$ (with $r, b \in \mathbb{Z}_p$, which is always possible), then the first three equations ensure that $\lambda_1 = r$, $\lambda_2 = b$ and $\lambda_3 = -rb$, while the last equation (right bottom) ensures that $b(b - 1) = 0$, i.e., $b \in \{0, 1\}$, as it holds that $(h^r g^b/g)^b h^{-rb} = g^{b(b-1)} = 1$.

Therefore, if we introduce the notation $\hat{\boldsymbol{C}} = \theta(C) := \begin{pmatrix} u & e & 1 & 1 \end{pmatrix} \in \mathbb{G}^4$, then the language $\mathscr{L}$ can be defined as the set of $C = (u, e)$ such that $\hat{\boldsymbol{C}}$ is in the subspace of $\mathbb{G}^4$ generated by the rows of the following matrix

$$\Gamma := \begin{pmatrix} g & h & 1 & 1 \\ 1 & g & u & e/g \\ 1 & 1 & g & h \end{pmatrix} . \qquad \square$$

Example 5 (Conjunction of Languages). Let g_i and h_i (for $i = 1, 2$) be four generators of $\mathbb{G}$, and $\mathscr{L}_i$ be (as in Example 3) the languages of DDH tuples in bases (g_i, h_i) respectively. We are now interested in the language $\mathscr{L} = \mathscr{L}_1 \times \mathscr{L}_2 \subseteq \mathbb{G}^4$, which is thus the conjunction of $\mathscr{L}_1 \times \mathbb{G}^2$ and $\mathbb{G}^2 \times \mathscr{L}_2$: it can be seen as the subspace of $\mathbb{G}^4$ generated by the rows of the following matrix

$$\Gamma := \begin{pmatrix} g_1 & h_1 & 1 & 1 \\ 1 & 1 & g_2 & h_2 \end{pmatrix} . \qquad \square$$

This can also be seen as the matrix, diagonal by blocks, with Γ_1 and Γ_2 the matrices for $\mathscr{L}_1$ and $\mathscr{L}_2$ respectively.

More formally, the generic framework for SPHFs in [7] considers the languages $\mathscr{L} \subseteq \mathcal{X}$ defined as follows: There exist two functions θ and Γ from the set of words $\mathcal{X}$ to the vector space $\mathbb{G}^n$ of dimension n, and to set $\mathbb{G}^{k \times n}$ of $k \times n$ matrices over $\mathbb{G}$, such that $C \in \mathscr{L}$ if and only if $\hat{\boldsymbol{C}} := \theta(C)$ is a linear combination of the rows of $\Gamma(C)$. From a witness w for a word C, it should be possible to compute such a linear combination as a row vector $\boldsymbol{\lambda} = (\lambda_i)_{i=1,\dots,k} \in \mathbb{Z}_p^{1 \times k}$:

$$\hat{\boldsymbol{C}} = \theta(C) = \boldsymbol{\lambda} \bullet \Gamma(C). \tag{1}$$

For the sake of simplicity, because of the equivalence between w and $\boldsymbol{\lambda}$, we will use them indifferently for the witness.

SPHFs. Let us now build an SPHF on such a language. A hashing key hk is just a random column vector $\mathsf{hk} \in \mathbb{Z}_p^n$, and the associated projection key is $\mathsf{hp} := \Gamma(C) \bullet \mathsf{hk}$. The hash value of a word C is then $H := \hat{\boldsymbol{C}} \bullet \mathsf{hk}$, and if $\boldsymbol{\lambda}$ is a witness for $C \in \mathscr{L}$, this hash value can also be computed as:

$$H = \hat{\boldsymbol{C}} \bullet \mathsf{hk} = \boldsymbol{\lambda} \bullet \Gamma(C) \bullet \mathsf{hk} = \boldsymbol{\lambda} \bullet \mathsf{hp} = \mathsf{proj}H,$$

which only depends on the witness $\boldsymbol{\lambda}$ and the projection key hp. On the other hand, if $C \notin \mathscr{L}$, then $\hat{\boldsymbol{C}}$ is linearly independent from the rows of $\Gamma(C)$. Hence, $H := \hat{\boldsymbol{C}} \bullet \mathsf{hk}$ looks random even given $\mathsf{hp} := \Gamma(C) \bullet \mathsf{hk}$, which is exactly the *smoothness* property.

Example 6. The SPHF corresponding to the language in Example 4, is then defined by:

$$\mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2, \mathsf{hk}_3, \mathsf{hk}_4)^\intercal \xleftarrow{\$} \mathbb{Z}_p^4$$
$$\mathsf{hp} = \Gamma(C) \bullet \mathsf{hk} = (g^{\mathsf{hk}_1} h^{\mathsf{hk}_2}, g^{\mathsf{hk}_2} u^{\mathsf{hk}_3} (e/g)^{\mathsf{hk}_4}, g^{\mathsf{hk}_3} h^{\mathsf{hk}_4})$$
$$H = \hat{\boldsymbol{C}} \bullet \mathsf{hk} = u^{\mathsf{hk}_1} e^{\mathsf{hk}_2} \qquad \mathsf{proj}H = \boldsymbol{\lambda} \bullet \mathsf{hp} = \mathsf{hp}_1^r \cdot \mathsf{hp}_2^b \cdot \mathsf{hp}_3^{-rb}.$$

For the sake of clarity, we will omit the C argument, and write Γ, instead of $\Gamma(C)$.

3.2 Limitations of Smooth Projective Hash Functions

At a first glance, as explained in the introduction, it may look possible to construct an iZK from an SPHF for the same language $\mathscr{L} = \mathsf{i}\mathscr{L}$ as follows:

- $\mathsf{iSetup}(\mathsf{crs})$ and $\mathsf{iTSetup}(\mathsf{crs})$ outputs the empty CRS $\mathsf{icrs} := \perp$;
- $\mathsf{iKG}(\mathsf{icrs}, \mathsf{x}, \mathsf{iw})$ outputs an empty public key $\mathsf{ipk} := \perp$ together with the secret key $\mathsf{isk} := (\mathsf{x}, \mathsf{iw})$;
- $\mathsf{iEnc}(\mathsf{ipk}, \mathsf{x})$ generates a random hashing key $\mathsf{hk} \overset{\$}{\leftarrow} \mathsf{HashKG}(\mathsf{crs}, \mathsf{x})$ and outputs the ciphertext $c := \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{crs}, \mathsf{x})$ together with the ephemeral key $K := H \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathsf{crs}, \mathsf{x})$;
- $\mathsf{iDec}(\mathsf{isk}, c)$ outputs the ephemeral key $K := \mathsf{proj}H \leftarrow \mathsf{ProjHash}(\mathsf{hp}, \mathsf{crs}, \mathsf{x}, \mathsf{iw})$.

This construction is sound: if $\mathsf{x} \notin \mathscr{L}$, given only $c = \mathsf{hp}$, the smoothness ensures that $K = H$ looks random. Unfortunately, there seems to be no way to compute K from only c, or in other words, there does not seem to exist algorithms iTKG and iTDec.

Example 6 is not Zero-Knowledge. Actually, with the SPHF from Example 6, no such algorithm iTKG or iTDec (verifying the zero-knowledge property) exists. It is even worse than that: a malicious verifier may get information about the witness, even if he just has a feedback whether the prover could use the correct hash value or not (and get the masked value or not), in a protocol such as the one in Fig. 1. A malicious verifier can indeed generate a ciphertext $c = \mathsf{hp}$, by generating hp_1 honestly but by picking hp_2 and hp_3 uniformly at random. Now, a honest prover will compute $\mathsf{proj}H = \mathsf{hp}_1^r \mathsf{hp}_2^b \mathsf{hp}_3^{-rb}$, to get back the ephemeral key (using iDec). When C is an encryption of $b = 1$, this value is random and independent of H, as hp_2 and hp_3 have been chosen at random, while when $b = 0$, this value is the correct $\mathsf{proj}H$ and is equal to H. Thus the projected hash value $\mathsf{proj}H$, which is the ephemeral output key by the honest prover, reveals some information about b, part of the witness.

If we want to avoid such an attack, the prover has to make sure that the hp he received was built correctly. Intuitively, this sounds exactly like the kind of verifications we could make with an SPHF: we could simply build an SPHF on the language of the "correctly built" hp. Then the prover could send a projection key for this new SPHF and ask the verifier to XOR the original hash value H with the hash value of this new SPHF. However, things are not that easy: first this does not solve the limitation due to the security proof (the impossibility of computing H for $\mathsf{x} \notin \mathsf{i}\mathscr{L}$) and second, in the SPHF in Example 6, all projection keys are valid (since Γ is full-rank, for any hp, there exists necessarily a hk such that $\mathsf{hp} = \Gamma \bullet \mathsf{hk}$).

3.3 iZK Construction

Let us consider an SPHF defined as in Section 3.1 for a language $\mathsf{i}\mathscr{L} = \mathscr{L}$. In this section, we show how to design, step by step, an iZK for $\mathsf{i}\mathscr{L}$ from this SPHF, following the overview in Section 1. At the end, we provide a summary of the construction and a complete proof. We illustrate our construction on the language of ElGamal ciphertexts of bits (Examples 4 and 6), and refer to this language as "our example". We suppose a cyclic group $\mathbb{G}$ of prime order p is fixed, and that DDH is hard in $\mathbb{G}$[5].

[5] The construction can be trivially extended to DLin, or any MDDH assumption [21] though.

We have seen the limitations of directly using the original SPHF are actually twofold. First, SPHFs do not provide a way to compute the hash value of a word outside the language, with just a projection key for which the hashing key is not known. Second, nothing ensures that a projection key has really been derived from an actually known hashing key, and in such a bad case, the projected hash value may leak some information about the word C (and the witness).

To better explain our construction, we first show how to overcome the first limitation. Thereafter, we will show how our approach additionally allows to check the validity of the projection keys (with a non-trivial validity meaning). It will indeed be quite important to notice that the projection keys coming from our construction (according to one of the setups) will not necessarily be valid (with a corresponding hashing key), as the corresponding matrix $\mathit{\Gamma}$ will not always be full rank, contrary to the projection keys of the SPHF in Example 6. Hence, the language of the valid projection keys will make sense in this setting.

Adding the Trapdoor. The CRS of our construction is a tuple $\mathsf{icrs} = (g', h', u' = {g'}^{r'}, e' = {h'}^{s'}) \in \mathbb{G}^4$, with g', h' two random generators of $\mathbb{G}$, and

- r', s' two random distinct scalars in $\mathbb{Z}_p$, for the normal CRS generated by iSetup, so that (g', h', u', e') is not a DDH tuple;
- $r' = s'$ a random scalar in $\mathbb{Z}_p$, for the trapdoor CRS generated by $\mathsf{iTSetup}$, with $\mathsf{i}\mathcal{T} = r'$ the trapdoor, so that (g', h', u', e') is a DDH tuple.

Then, we build an SPHF for the augmented language $\mathscr{L}_t$ defined as follows: a word $C_t = (C, u', e')$ is in $\mathscr{L}_t$ if and only if either C is in the original language $\mathscr{L}$ or (u', e') is a DDH tuple. This new language $\mathscr{L}_t$ can be seen as the disjunction of the original language $\mathscr{L}$ and of the DDH language in basis (g', h'). Construction of disjunctions of SPHFs were proposed in [2] but require pairings. In this article, we use an alternative more efficient construction without pairing[6]. Let us show it on our example, with $C_t = (C, u', e')$. We set $\hat{\boldsymbol{C}}_t \coloneqq ({g'}^{-1}, 1, 1, 1, 1, 1, 1)$ and $\mathit{\Gamma}_t(C_t) \in \mathbb{G}^{(k+3)\times(n+3)}$ as

$$\mathit{\Gamma}_t(C_t) \coloneqq \left(\begin{array}{ccc|ccc} \multicolumn{3}{c|}{\mathbf{1}} & \multicolumn{3}{c}{\mathit{\Gamma}(C)} \\ \hline g' & 1 & 1 & \multicolumn{3}{c}{\hat{\boldsymbol{C}} = \theta(C)} \\ \hline 1 & g' & h' & 1 & \dots & 1 \\ \hline g' & u' & e' & 1 & \dots & 1 \end{array}\right) = \left(\begin{array}{ccc|cccc} 1 & 1 & 1 & g & h & 1 & 1 \\ 1 & 1 & 1 & 1 & g & u & e/g \\ 1 & 1 & 1 & 1 & 1 & g & h \\ \hline g' & 1 & 1 & u & e & 1 & 1 \\ \hline 1 & g' & h' & 1 & 1 & 1 & 1 \\ \hline g' & u' & e' & 1 & 1 & 1 & 1 \end{array}\right). \quad (2)$$

Let us show the language corresponding to $\mathit{\Gamma}_t$ and $\hat{\boldsymbol{C}}_t$ is indeed $\mathscr{L}_t$: Due to the first column of $\mathit{\Gamma}_t$ and the first element of $\hat{\boldsymbol{C}}_t$, if $\hat{\boldsymbol{C}}_t$ is a linear combination of rows of $\mathit{\Gamma}_t$ with coefficients $\boldsymbol{\lambda_t}$ (i.e., $\hat{\boldsymbol{C}}_t = \boldsymbol{\lambda_t} \bullet \mathit{\Gamma}_t$), one has $\lambda_{t,4} + \lambda_{t,6} = -1$, and thus at least $\lambda_{t,4}$ or $\lambda_{t,6}$ is not equal to zero.

[6] Contrary to [2] however, our matrix $\mathit{\Gamma}_t$ depends on the words C_t, which is why we get this more efficient construction.

– If $\lambda_{t,6} \neq 0$, looking at the second and the third columns of Γ_t gives that:

$$\lambda_{t,5} \bullet (g',\ h') + \lambda_{t,6} \bullet (u',\ e') = (1,1)\,, \text{i.e.,}\ (u',e') = (g'^{\lambda_{t,5}/\lambda_{t,6}},\ h'^{\lambda_{t,5}/\lambda_{t,6}}),$$

or in other words (u',e') is a DDH tuple in basis (g',h');

– if $\lambda_{t,4} \neq 0$, looking at the last four columns of Γ_t gives that: $\lambda_{t,4} \bullet \hat{\boldsymbol{C}} = \lambda_{t,4} \bullet (u,e,1,1)$ is a linear combination of rows of Γ, hence $\hat{\boldsymbol{C}}$ too. As a consequence, by definition of $\mathscr{L}$, $C \in \mathscr{L}$.

Now, whatever the way the CRS is generated (whether (u',e') is a DDH tuple or not), it is always possible to compute $\mathsf{proj}H$ as follows, for a word $C \in \mathscr{L}$ with witnesses r and b:

$$\mathsf{proj}H = \boldsymbol{\lambda_t} \bullet \mathsf{hp} \qquad\qquad \boldsymbol{\lambda_t} = (\boldsymbol{\lambda}, -1, 0, 0) = (r, b, -rb, -1, 0, 0)$$

When the CRS is generated with the normal setup, as shown above, this is actually the only way to compute $\mathsf{proj}H$, since (u',e') is not a DDH tuple and so $\hat{\boldsymbol{C}}_t$ is linearly dependent of the rows of Γ_t if and only if $C \in \mathscr{L}$. On the opposite, when the CRS is generated by the trapdoor setup with trapdoor r', we can also compute $\mathsf{proj}H$ using the witness r': $\mathsf{proj}H = \boldsymbol{\lambda}'_{\boldsymbol{t}} \bullet \mathsf{hp}$ with $\boldsymbol{\lambda}'_{\boldsymbol{t}} = (0,0,0,0,r',-1)$.

However, the latter way to compute $\mathsf{proj}H$ gives the same result as the former way, only if $\mathsf{hp}_{t,5}$ and $\mathsf{hp}_{t,6}$ involve the correct value for hk_1. A malicious verifier could decide to choose random $\mathsf{hp}_{t,5}$ and $\mathsf{hp}_{t,6}$, which would make $\boldsymbol{\lambda}'_{\boldsymbol{t}} \bullet \mathsf{hp}$ look random and independent of the real hash value!

Ensuring the Validity of Projection Keys. The above construction and trapdoor would provide zero-knowledge if we could ensure that the projection keys hp (generated by a potentially malicious verifier) is valid, so that, intuitively, $\mathsf{hp}_{t,5}$ and $\mathsf{hp}_{t,6}$ involve the correct value of hk_1. Using a zero-knowledge proof (that hp derives from some hashing key hk) for that purpose would annihilate all our efforts to avoid adding rounds and to work under plain DDH (interactive ZK proofs introduce more rounds, and Groth-Sahai [32] NIZK would require assumptions on bilinear groups). So we are left with doing the validity check again with SPHFs.

Fortunately, the language of valid projection keys hp can be handled by the generic framework, since a valid projection key hp is such that: $\mathsf{hp} = \Gamma_t \bullet \mathsf{hk}$, or in other words, if we transpose everything $\mathsf{hp}^\intercal = \mathsf{hk}^\intercal \bullet \Gamma_t^\intercal$. This is exactly the same as in Equation (1), with $\hat{\boldsymbol{C}} \leftrightarrow \mathsf{hp}^\intercal$, $\Gamma \leftrightarrow \Gamma_t^\intercal$ and witness $\boldsymbol{\lambda} \leftrightarrow \mathsf{hk}^\intercal$. So we can now define a smooth projective hash function on that language, where the projection key is called transposed projection key tp, the hashing key is called transposed hashing key tk, the hash value is called transposed hash value $\mathsf{t}H$ and the projected hash value is called transposed projected hash value $\mathsf{tproj}H$.

Finally, we could define an iZK, similarly to the one in Section 3.2, except, ipk contains a transposed projection key tp (generated by the prover from a random transposed hashing key tk), and c contains the associated transposed projected hash value $\mathsf{tproj}H$ in addition to hp, so that the prover can check using tk that hp is valid by verifying whether $\mathsf{tproj}H = \mathsf{t}H$ or not.

An Additional Step. Unfortunately, we are not done yet, as the above modification breaks the soundness property! Indeed, in this last construction, the prover now learns an additional information about the hash value H: $\mathsf{tproj}H = \mathsf{hk}^\intercal \mathsf{tp}$, which does depend on the secret key hk. He could therefore choose $\mathsf{tp} = \hat{\boldsymbol{C}}_t^\intercal$, so that $\mathsf{tproj}H = \mathsf{hk}^\intercal \hat{\boldsymbol{C}}_t^\intercal = \hat{\boldsymbol{C}}_t \mathsf{hk}$ is the hash value $H = K$ of C under hk.

We can fix this by ensuring that the prover will not know the extended word $\hat{\boldsymbol{C}}_t$ on which the SPHF will be based when he sends tp, using an idea similar to the 2-universality property of SPHF introduced by Cramer and Shoup in [18]. For that purpose, we extend Γ_t and make $\hat{\boldsymbol{C}}_t$ depends on a random scalar $\zeta \in \mathbb{Z}_p$ chosen by the verifier (and included in c).

Detailed Construction. Let us now formally show how to build an iZK from any SPHF built from the generic framework of [7], following the previous ideas. We recall that we consider a language $\mathscr{L} = \mathsf{i}\mathscr{L}$, such that a word $\mathsf{x} = C$ is in $\mathsf{i}\mathscr{L}$, if and only if $\hat{\boldsymbol{C}} = \theta(C)$ is a linear combination of the rows of some matrix $\Gamma \in \mathbb{G}^{k \times n}$ (which may depend on C). The coefficients of this linear combination are entries of a row vector $\boldsymbol{\lambda} \in \mathbb{Z}_p^{1 \times k}$: $\hat{\boldsymbol{C}} = \boldsymbol{\lambda} \bullet \Gamma$, where $\boldsymbol{\lambda} = \boldsymbol{\lambda}(\mathsf{iw})$ can be computed from the witness iw for x.

The setup algorithms $\mathsf{iSetup}(\mathsf{crs})$ and $\mathsf{iTSetup}(\mathsf{crs})$ are defined as above (page 13). We define an extended language using the generic framework:

$$\theta_t(\mathsf{x}, \zeta) = \hat{\boldsymbol{C}}_t = (g'^{-1}, 1, \dots, 1, g'^{-\zeta}, 1, \dots, 1) \qquad \in \mathbb{G}^{1 \times (2n+6)}$$

$$\Gamma_t(\mathsf{x}) = \left(\begin{array}{c|c} \Gamma'_t(\mathsf{x}) & \mathbf{1} \\ \hline \mathbf{1} & \Gamma'_t(\mathsf{x}) \end{array}\right) \qquad \in \mathbb{G}^{(2k+6) \times (2n+6)},$$

where $\Gamma'_t(\mathsf{x})$ is the matrix (initially called $\Gamma_t(\mathsf{x})$ in Equation (2), $\mathbf{1}$ is the matrix of $\mathbb{G}^{(2k+3) \times (2n+3)}$ with all entries equal to 1, and ζ is a scalar used to ensure the prover cannot guess the word $\hat{\boldsymbol{C}}_t$ which will be used, and so cannot choose $\mathsf{tp} = \hat{\boldsymbol{C}}_t$. As explained above, this language corresponds to a 2-universal SPHF for the disjunction of the language of DDH tuples (g', h', u', e') and the original language $\mathscr{L}$. We write:

$$\begin{aligned} \boldsymbol{\lambda_t}(\zeta, \mathsf{iw}) &= (\boldsymbol{\lambda}(\mathsf{iw}), -1, 0, 0, \zeta\boldsymbol{\lambda}(\mathsf{iw}), -\zeta, 0, 0) \\ \boldsymbol{\lambda_t}(\zeta, \mathsf{i}\mathcal{T}) &= (0, \dots, 0, r', -1, 0, \dots, 0, \zeta r', -\zeta) \qquad \text{with } \mathsf{i}\mathcal{T} = r', \end{aligned}$$

so that:

$$\hat{\boldsymbol{C}}_t = \begin{cases} \boldsymbol{\lambda_t}(\zeta, \mathsf{iw}) \bullet \Gamma_t(\mathsf{x}) & \text{if } (g', h', u', e') \text{ is a DDH tuple, with witness } \mathsf{i}\mathcal{T} \\ \boldsymbol{\lambda_t}(\zeta, \mathsf{i}\mathcal{T}) \bullet \Gamma_t(\mathsf{x}) & \text{if } \mathsf{x} \in \mathsf{i}\mathscr{L} \text{ with witness } \mathsf{iw}. \end{cases}$$

The resulting iZK construction is depicted in Fig. 4. This is a slightly more efficient construction that the one we sketched previously, where the prover does not test anymore explicitly $\mathsf{tproj}H$, but $\mathsf{tproj}H$ (or $\mathsf{t}H$) is used to mask K. Thus, $\mathsf{tproj}H$ no more needs to be included in c.

Variants. In numerous cases, it is possible to add the trapdoor in a slightly more efficient way, if we accept to use word-dependent CRS. While the previous construction would be useful for security in the UC framework [15], the more efficient construction with a word-dependent CRS is enough in the stand-alone setting. Independently of that improvement, it is also possible to slightly reduce the size of hp, by computing ζ with an entropy extractor, and so dropping it from hp. Details for both variants are given in the full version [8].

$\underline{\mathsf{iSetup}(\mathsf{crs})}$ $(g', h') \xleftarrow{\$} \mathbb{G}^{*2}$ $(r', s') \xleftarrow{\$} \mathbb{Z}_p^2 \setminus \{(a, a) \mid a \in \mathbb{Z}_p\}$ $(u', e') \leftarrow ({g'}^{r'}, {h'}^{s'}) \in \mathbb{G}^2$ $\mathsf{icrs} \leftarrow (g', h', u', e')$ **return** icrs	$\underline{\mathsf{iTSetup}(\mathsf{crs})}$ $(g', h') \xleftarrow{\$} \mathbb{G}^{*2}$ $r' \xleftarrow{\$} \mathbb{Z}_p$ $(u', e') \leftarrow ({g'}^{r'}, {h'}^{r'}) \in \mathbb{G}^2$ $\mathsf{icrs} \leftarrow (g', h', u', e')$; $\mathsf{i}\mathcal{T} \leftarrow r'$ **return** $(\mathsf{icrs}, \mathsf{i}\mathcal{T})$
$\underline{\mathsf{iKG}(\mathsf{icrs}, \mathsf{x}, \mathsf{iw})}$ $\mathsf{tk} \xleftarrow{\$} \mathbb{Z}_p^{2k+6}$ $\mathsf{ipk} := \mathsf{tp} \leftarrow \Gamma_t(\mathsf{x})^{\intercal} \bullet \mathsf{tk} \in \mathbb{G}^{2n+6}$ $\mathsf{isk} := (\mathsf{x}, \mathsf{tk}, \mathsf{iw})$ **return** $(\mathsf{ipk}, \mathsf{isk})$	$\underline{\mathsf{iTKG}(\mathsf{icrs}, \mathsf{x}, \mathsf{i}\mathcal{T})}$ $\mathsf{tk} \xleftarrow{\$} \mathbb{Z}_p^{2k+6}$ $\mathsf{ipk} := \mathsf{tp} \leftarrow \Gamma_t(\mathsf{x})^{\intercal} \bullet \mathsf{tk} \in \mathbb{G}^{2n+6}$ $\mathsf{itk} := (\mathsf{x}, \mathsf{tk}, \mathsf{i}\mathcal{T})$ **return** $(\mathsf{ipk}, \mathsf{itk})$
$\underline{\mathsf{iEnc}(\mathsf{icrs}, \mathsf{ipk}, \mathsf{x})}$ $\mathsf{tp} \leftarrow \mathsf{ipk}$; $\mathsf{hk} \xleftarrow{\$} \mathbb{Z}_p^{2n+6}$; $\zeta \xleftarrow{\$} \mathbb{Z}_p$ $\mathsf{hp} \leftarrow \Gamma_t(\mathsf{x}) \bullet \mathsf{hk} \in \mathbb{Z}_p^{2k+6}$ $\mathsf{tproj}H \leftarrow \mathsf{hk}^{\intercal} \bullet \mathsf{tp} \in \mathbb{G}$	$H \leftarrow \theta_t(\mathsf{x}, \zeta) \bullet \mathsf{hk} \in \mathbb{Z}_p$ $K \leftarrow H \cdot \mathsf{tproj}H \in \mathbb{G}$ $c := (\zeta, \mathsf{hp})$ **return** (K, c)
$\underline{\mathsf{iDec}(\mathsf{icrs}, \mathsf{isk}, c)}$ $(\mathsf{x}, \mathsf{tk}, \mathsf{iw}) \leftarrow \mathsf{isk}$ $(\zeta, \mathsf{hp}) \leftarrow c$ $\mathsf{t}H \leftarrow \mathsf{hp}^{\intercal} \bullet \mathsf{tk} \in \mathbb{Z}_p$ $\mathsf{proj}H \leftarrow \boldsymbol{\lambda_t}(\zeta, \mathsf{iw}) \bullet \mathsf{hp} \in \mathbb{G}$ **return** $K := \mathsf{proj}H \cdot \mathsf{t}H \in \mathbb{G}$	$\underline{\mathsf{iTDec}(\mathsf{icrs}, \mathsf{itk}, c)}$ $(\mathsf{x}, \mathsf{tk}, \mathsf{i}\mathcal{T}) \leftarrow \mathsf{itk}$ $(\zeta, \mathsf{hp}) \leftarrow c$ $\mathsf{t}H \leftarrow \mathsf{hp}^{\intercal} \bullet \mathsf{tk} \in \mathbb{Z}_p$ $\mathsf{trap}H := \boldsymbol{\lambda_t}(\zeta, \mathsf{i}\mathcal{T}) \bullet \mathsf{hp} \in \mathbb{G}$ **return** $K := \mathsf{trap}H \cdot \mathsf{t}H \in \mathbb{G}$

Fig. 4: Construction of iZK

3.4 SSiZK Construction

Our SSiZK construction is similar to our iZK construction, except that, in addition both iSetup and iTSetup add the CRS icrs, a tuple $(v_{k,i})_{i=0,\ldots,2\mathfrak{K}}^{k=1,2}$ of group elements constructed as follows: for $i = 0$ to $2\mathfrak{K}$ (with $\mathfrak{K}$ the security parameter): $r'_i \xleftarrow{\$} \mathbb{Z}_p$, $v_{1,i} \leftarrow {g'}^{r'_i}$, $v_{2,i} \leftarrow {h'}^{r'_i}$. We also define the two Waters functions [45] $\mathcal{W}_k : \{0,1\}^{2\mathfrak{K}} \to \mathbb{G}$, as $\mathcal{W}_k(m) = v_{k,0} \prod_{i=1}^{2\mathfrak{K}} v_{k,i}^{m_i}$, for any bitstring $m = m_1 \| \ldots \| m_{2\mathfrak{K}} \in \{0,1\}^{2\mathfrak{K}}$. Finally, the CRS is also supposed to contain a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{2\mathfrak{K}}$ drawn from a collision-resistant hash function family $\mathcal{HF}$.

Next, the language $\mathscr{L}_t$ is further extended by adding 3 rows and 2 columns (all equal to 1 except on the 3 new rows) to both the sub-matrices $\Gamma'_t(\mathsf{x})$ of $\Gamma_t(\mathsf{x})$, where the 3 new rows are:

$$\left(\begin{array}{c|cc|ccc|cc} 1 & 1 & 1 & 1 & \ldots & 1 & g' & h' \\ 1 & 1 & 1 & 1 & \ldots & 1 & u'' & e'' \\ \hline g' & 1 & 1 & 1 & \ldots & 1 & g' & 1 \end{array}\right) \in \mathbb{G}^{3 \times (n+5)},$$

with $u'' = \mathcal{W}_1(\mathcal{H}(\ell, \mathsf{x}))$ and $e'' = \mathcal{W}_2(\mathcal{H}(\ell, \mathsf{x}))$. The vector $\hat{\boldsymbol{C}}_t$ becomes $\hat{\boldsymbol{C}}_t = (g^{-1}, 1, \ldots, 1, g^{-\zeta}, 1, \ldots, 1)$ (it is the same except for the number of 1's). Due to lack of space, the full matrix is depicted in the full version [8], where the security proof can also be found. The security proof requires that $\mathsf{Setup}_{\mathsf{crs}}$ also outputs some additional information or trapdoor $\mathcal{T}_{\mathsf{crs}}$, which enables to check, in polynomial time, whether a given word x is in $\mathrm{i}\mathscr{L}$ or not.

Here is an overview of the security proof. Correctness, setup indistinguishability, and zero-knowledge are straightforward. Soundness follows from the fact that (g', h', u'', e'') is a DDH-tuple, when parameters are generated by iSetup (and also $\mathsf{iTSetup}$ actually), and so $(g', 1)$ is never in the subspace generated by (g', h') and (u'', e'') (as $h' \neq 1$), hence the corresponding language $\mathscr{L}_t$ is the same as for our iZK construction. Finally, to prove simulation-soundness, we use the programmability of the Waters function [33] and change the generation of the group elements $(v_{k,i})$ so that for the challenge proof (generated by the adversary) (g', h', u'', e'') is not a DDH-tuple, while for the simulated proofs it is a DDH-tuple. Then, we can change the setup to iSetup, while still being able to simulate proofs. But in this setting, the word $\hat{\boldsymbol{C}}_t$ for the challenge proof is no more in $\mathscr{L}_t$, and smoothness implies simulation-soundness.

4 Application to the Inner Product

In case of biometric authentication, a server $\mathcal{S}$ wants to compute the Hamming distance between a fresh user's feature and the stored template, but without asking the two players to reveal their own input: the template y from the server side and the fresh feature x from the client side. One can see that the Hamming distance between the ℓ-bit vectors x and y is the sum of the Hamming weights of x and y, minus twice the inner product of x and y. Let us thus focus on this private evaluation of the inner product: a client $\mathcal{C}$ has an input $x = (x_i)_{i=1}^{\ell} \in \{0,1\}^{\ell}$ and a server $\mathcal{S}$ has an input $y = (y_i)_{i=1}^{\ell} \in \{0,1\}^{\ell}$. The server $\mathcal{S}$ wants to learn the inner product $\mathsf{IP} = \sum_{i=1}^{\ell} x_i y_i \in \{0, \dots, \ell\}$, but nothing else, while the client $\mathcal{C}$ just learns whether the protocol succeeded or was aborted.

Semi-Honest Protocol. $\mathcal{C}$ can send an ElGamal encryption of each bit under a public key of her choice and then $\mathcal{S}$ can compute an encryption of $\mathsf{IP} + R$, with $R \in \mathbb{Z}_p$ a random mask, using the homomorphic properties of ElGamal, and sends this ciphertext. $\mathcal{C}$ finally decrypts and sends back $g^{\mathsf{IP}+R}$ to $\mathcal{S}$ who divides it by g^R to get g^{IP}. Since IP is small, an easy discrete logarithm computation leads to IP.

Malicious Setting. To transform this semi-honest protocol into one secure against malicious adversaries, we could apply our generic conversion presented in the full version [8]. Here, we propose an optimized version of this transformation for this protocol. We use the ElGamal scheme for the encryption $\mathcal{E}_{\mathsf{pk}}$, where pk is a public key chosen by $\mathcal{C}$ and the secret key is $\mathsf{sk} = (\mathsf{sk}_j)_{j=1}^{\log p}$, and the Cramer-Shoup scheme [17] for commitments Com, of group elements or multiple group elements with randomness reuse, where the public key is in the CRS. The CRS additionally contains the description of a cyclic group and a generator g of this group. The construction is presented on Figure 5. First, the client commits to her secret key (this is the most efficient alternative as soon as $n \gg \ell$) and sends encryptions $(c_i)_{i \leq n}$ of her bits. Then, the server commits to his inputs $(y_i)_i$ and to two random integers (R, R'), computes the encryption $(\hat{u}, \hat{e})$ of $g^{R \cdot \mathsf{IP} + R'}$), re-randomized with a randomness ρ, masked by an iZK to ensure that the c_i's encrypt bits under the key pk whose corresponding secret key sk is committed (masking one of the two components of an ElGamal ciphertext suffices). The client

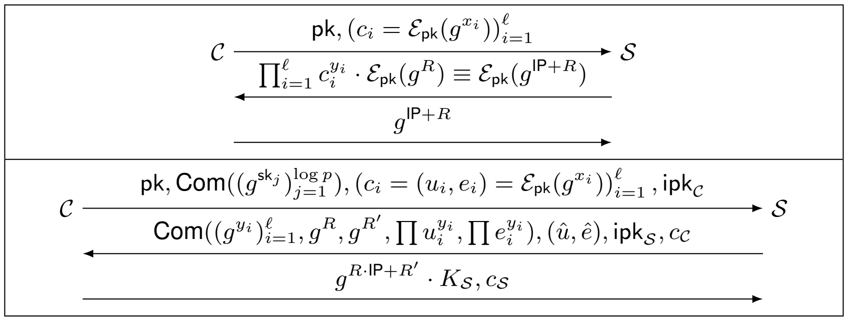

$\mathcal{C}$
$\mathcal{S}$
$\mathsf{pk}, (c_i = \mathcal{E}_{\mathsf{pk}}(g^{x_i}))_{i=1}^{\ell}$
$\prod_{i=1}^{\ell} c_i^{y_i} \cdot \mathcal{E}_{\mathsf{pk}}(g^R) \equiv \mathcal{E}_{\mathsf{pk}}(g^{\mathsf{IP}+R})$
$g^{\mathsf{IP}+R}$
$\mathcal{C}$
$\mathcal{S}$
$\mathsf{pk}, \mathsf{Com}((g^{\mathsf{sk}_j})_{j=1}^{\log p}), (c_i = (u_i, e_i) = \mathcal{E}_{\mathsf{pk}}(g^{x_i}))_{i=1}^{\ell}, \mathsf{ipk}_{\mathcal{C}}$
$\mathsf{Com}((g^{y_i})_{i=1}^{\ell}, g^R, g^{R'}, \prod u_i^{y_i}, \prod e_i^{y_i}), (\hat{u}, \hat{e}), \mathsf{ipk}_{\mathcal{S}}, c_{\mathcal{C}}$
$g^{R \cdot \mathsf{IP}+R'} \cdot K_{\mathcal{S}}, c_{\mathcal{S}}$

Fig. 5: Semi-Honest and Malicious Protocols for Secure Inner Product Computation

replies with $g^{R \cdot \mathsf{IP}+R'}$, masked by a SSiZK (this is required for UC security) to ensure that the $\mathsf{Com}(g^{y_i})$ contains bits, and that the masked ciphertext has been properly built. The server then recovers $g^{R \cdot \mathsf{IP}+R'}$, removes R and R', and tries to extract the discrete logarithm IP. If no solution exists in $\{0, \ldots, \ell\}$, the server aborts. This last verification avoids the 2-round verification phase from our generic compiler: if the client tries to cheat on $R \cdot \mathsf{IP} + R'$, after removing R and R', the result would be random, and thus in the appropriate range with negligible probability ℓ/p, since ℓ is polynomial and p is exponential. We prove in the full version [8] that *the above protocol is secure against malicious adversaries in the UC framework with static corruptions, under the plain DDH assumption, and in the common reference string setting*.

Efficiency and Comparison with Other Methodologies. In the full version [8], we provide a detailed analysis of our inner product protocol in terms of complexity. Then, we estimate the complexity of this protocol when, instead of using iZK, the security against malicious adversaries in the UC model is ensured by using the Groth-Sahai methodology [32] or Σ-protocols. In this section, we sum up our comparisons in a table. The notation $>$ indicates that the given complexity is a lower bound on the real complexity of the protocol (we have not taken into account the linear blow-up incurred by the conversion of NIZK into SS-NIZK), and $\gg$ indicates a very loose lower bound. We stress that with usual parameter, an element of $\mathbb{G}_2$ is twice as big as an element of $\mathbb{G}_1$ (or $\mathbb{G}$) and the number of rounds in the major efficiency drawback (see Section 1). The efficiency improvement of iZK compared to NIZK essentially comes from their "batch-friendly" nature.

Proofs	Pairings	Exponentiations	Communication	Rounds
Σ-proofs	0	38ℓ	20ℓ	5
GS proofs	$> 14\ell$	$\gg 28\ell(\mathbb{G}_1) + 6\ell(\mathbb{G}_2)$	$> 11\ell(\mathbb{G}_1) + 10\ell(\mathbb{G}_2)$	3
iZK (this paper)	0	67ℓ	21ℓ	3

Moreover, our iZKs do not require pairings, which allows us to use more efficient elliptic curves than the best existing curves for the Groth-Sahai methodology. With a reasonable choice of two curves, one without pairing and one with pairing, for 128 bits of security, we get the following results: (counting efficiency as a multiple of the running time of an exponentiation in $\mathbb{G}_1$)

Curve \ Efficiency	Pairings	Exponentiations in $\mathbb{G}_1$	Exponentiations in $\mathbb{G}_2$
Curve25519 [10]	no pairings	1	✗
[11]	≈ 8	≈ 3	≈ 6

Acknowledgments. This work was supported in part by the CFM Foundation, ANR-14-CE28-0003 (Project EnBid), and the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud).

References

1. Abdalla, M., Benhamouda, F., Blazy, O., Chevalier, C., Pointcheval, D.: SPHF-friendly non-interactive commitments. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 214–234. Springer (Dec 2013)
2. Abdalla, M., Benhamouda, F., Pointcheval, D.: Disjunctions for hash proof systems: New constructions and applications. Cryptology ePrint Archive, Report 2014/483 (2014), http://eprint.iacr.org/2014/483
3. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: How to sell digital goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer (May 2001)
4. Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 1–16. Springer (May 2014)
5. Bellare, M., Boldyreva, A., Palacio, A.: An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 171–188. Springer (May 2004)
6. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Ashby, V. (ed.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993)
7. Benhamouda, F., Blazy, O., Chevalier, C., Pointcheval, D., Vergnaud, D.: New techniques for SPHFs and efficient one-round PAKE protocols. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 449–475. Springer (Aug 2013)
8. Benhamouda, F., Couteau, G., Pointcheval, D., Wee, H.: Implicit zero-knowledge arguments and applications to the malicious setting. Cryptology ePrint Archive, Report 2015/246 (2015), http://eprint.iacr.org/2015/246
9. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 626–643. Springer (Dec 2012)
10. Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer (Apr 2006)
11. Beuchat, J.L., González-Díaz, J.E., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal Ate pairing over Barreto-Naehrig curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) PAIRING 2010. LNCS, vol. 6487, pp. 21–39. Springer (Dec 2010)
12. Blazy, O., Pointcheval, D., Vergnaud, D.: Round-optimal privacy-preserving protocols with smooth projective hash functions. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 94–111. Springer (Mar 2012)
13. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. 37(2), 156–189 (1988), http://dx.doi.org/10.1016/0022-0000(88)90005-0

14. Brodkin, J.: Satellite internet faster than advertised, but latency still awful (Feb 2013), http://arstechnica.com/information-technology/2013/02/satellite-internet-faster-than-advertised-but-latency
15. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001)
16. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM 51(4), 557–594 (Jul 2004), http://doi.acm.org/10.1145/1008731.1008734
17. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 13–25. Springer (Aug 1998)
18. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 45–64. Springer (Apr / May 2002)
19. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography (extended abstract). In: 23rd ACM STOC. pp. 542–552. ACM Press (May 1991)
20. ECRYPT II: eBATS, http://bench.cr.yp.to/results-dh.html
21. Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for Diffie-Hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 129–147. Springer (Aug 2013)
22. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In: 31st FOCS. pp. 308–317. IEEE Computer Society Press (Oct 1990)
23. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer (Aug 1987)
24. Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 467–476. ACM Press (Jun 2013)
25. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 524–543. Springer (May 2003), http://eprint.iacr.org/2003/032.ps.gz
26. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. ACM Transactions on Information and System Security 9(2), 181–234 (2006)
27. Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting data privacy in private information retrieval schemes. In: 30th ACM STOC. pp. 151–160. ACM Press (May 1998)
28. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987)
29. Goldreich, O., Micali, S., Wigderson, A.: How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 171–185. Springer (Aug 1987)
30. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. 18(1), 186–208 (1989)
31. Granger, R., Kleinjung, T., Zumbrägel, J.: Breaking '128-bit secure' supersingular binary curves - (or how to solve discrete logarithms in $F_{2^{4 \cdot 1223}}$ and $F_{2^{12 \cdot 367}}$). In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 126–145. Springer (Aug 2014)
32. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer (Apr 2008)
33. Hofheinz, D., Kiltz, E.: Programmable hash functions and their applications. Journal of Cryptology 25(3), 484–527 (Jul 2012)

34. Huang, Y., Katz, J., Evans, D.: Efficient secure two-party computation using symmetric cut-and-choose. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 18–35. Springer (Aug 2013)
35. Ishai, Y., Kushilevitz, E., Lindell, Y., Petrank, E.: Black-box constructions for secure computation. In: Kleinberg, J.M. (ed.) 38th ACM STOC. pp. 99–108. ACM Press (May 2006)
36. Jarecki, S.: Practical covert authentication. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 611–629. Springer (Mar 2014)
37. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 1–17. Springer (Aug 2013)
38. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer (May 2007)
39. Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 329–346. Springer (Mar 2011)
40. Malkin, T., Teranishi, I., Vahlis, Y., Yung, M.: Signatures resilient to continual leakage on memory and computation. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 89–106. Springer (Mar 2011)
41. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: 22nd ACM STOC. pp. 427–437. ACM Press (May 1990)
42. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer (Aug 1990)
43. shelat, a., Shen, C.H.: Two-output secure computation with malicious adversaries. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 386–405. Springer (May 2011)
44. shelat, a., Shen, C.H.: Fast two-party secure computation with minimal assumptions. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 13. pp. 523–534. ACM Press (Nov 2013)
45. Waters, B.R.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer (May 2005)
46. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press (Oct 1986)