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

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 [GMR89]. 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 [Yao86, GMW87b, GMW87a], and as a tool for building cryptographic primitives with strong security guarantees such as encryption secure against chosen-ciphertext attacks [NY90, DDN91].

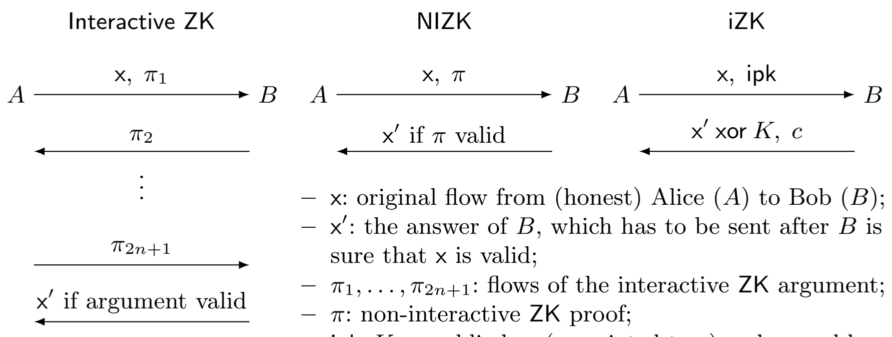
In this work, we focus on the use of zero-knowledge arguments as used in efficient two-party protocols for enforcing semi-honest behavior. We are particularly interested in round-efficient two-party protocols, as network latency and round-trip times can be a major efficiency bottleneck, for instance, 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 [GS08] and thus stronger assumptions and additional efficiency overhead, or the use of random oracles [BR93, FS87].

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" [Sch90] 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 [BPW12], 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 [Bro13]). Concretely, using Curve25519 elliptic curve of Bernstein [Ber06] (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.

[1] According to [II], 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)

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 [GS08], and we also avoid the theoretical pitfalls associated with instantiating the random oracle methodology [CGH04, BBP04]. From a practical stand-point, we can instantiate our DDH-based protocols over a larger classe of groups. Concrete examples include Bernstein's Curve25519 [Ber06] 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. 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).

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*).

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) [CS02]. 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 [GL06], 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 [GL03, GL06, BPV12, ABB+13].) 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 [BBC+13] 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 [FLS90]. 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.

[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.

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 [CS02].

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. [MTVY11]): 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 [Wat05], $\mathcal{W}_k(m) = v_{k,0} \prod_{i=1}^{|m|} v_{k,i}^{m_i}$, when $m = m_1 \| \dots \| m_{|m|}$ is a bitstring, the $v_{k,0}, \dots, v_{k,|m|}$ 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 [BBC+13], 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 [GS08] and the approach based on zero-knowledge proofs "à la Schnorr" [Sch90] in Appendix A. In addition, as proof of concept, we show in Appendix B 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 [GMW87b, GMW87a]. 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 Appendix F, 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 [ABP14]. However, the methods used in our paper are very different from the one in [ABP14], 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 [IKLP06, LP07, LP11, sS11, sS13, Lin13, HKE13] 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 [GS08], 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 [AIR01]. They realize implicit proofs using conditional disclosure of secrets [GIKM98]. The latter, together with witness encryption [GGSW13] and SPHFs, only provide a weak "honest-verifier zero-knowledge" guarantee.

Witness encryption was introduced by Garg et al. in [GGSW13]. 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, $X \overset{\$}{\leftarrow} \mathscr{X}$ indicates that X has been chosen uniformly at random in the (finite) set $\mathscr{X}$. 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 w;

[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 Appendix C.2.

- $(\mathsf{ipk}, \mathsf{itk}) \overset{\$}{\leftarrow} \mathsf{iTKG}^{\ell}(\mathsf{icrs}, i\mathcal{T}, \mathsf{x})$ 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.

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 $i\mathcal{L}$ (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 $(\mathsf{icrs}, i\mathcal{T}) \overset{\$}{\leftarrow} \mathsf{iTSetup}(\mathsf{crs})$, and for any $\mathsf{x} \in \mathcal{X}$ and any $\ell \in \{0,1\}^*$,
  - if $\mathsf{x} \in i\mathcal{L}$ with witness iw, $(\mathsf{ipk}, \mathsf{isk}) \overset{\$}{\leftarrow} \mathsf{iKG}^{\ell}(\mathsf{icrs}, \mathsf{x}, w)$, and $(c, K) \overset{\$}{\leftarrow} \mathsf{iEnc}^{\ell}(\mathsf{ipk}, \mathsf{x})$, then $K = \mathsf{iDec}^{\ell}(\mathsf{isk}, c)$;
  - if $(\mathsf{ipk}, \mathsf{itk}) \overset{\$}{\leftarrow} \mathsf{iTKG}^{\ell}(i\mathcal{T}, \mathsf{x})$ and $(c, K) \overset{\$}{\leftarrow} \mathsf{iEnc}^{\ell}(\mathsf{ipk}, \mathsf{x})$, then $K = \mathsf{iTDec}^{\ell}(\mathsf{itk}, c)$.
- **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:

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

- **Soundness.** When the CRS is generated as $\mathsf{icrs} \overset{\$}{\leftarrow} \mathsf{iSetup}(\mathsf{crs})$, and when $\mathsf{x} \notin \mathcal{L}$, 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 $\mathsf{x} \notin i\mathcal{L}$, for any label $\ell \in \{0,1\}^*$, the two distributions:

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

  are statistically indistinguishable (iEnc may output $(\bot, K)$ when the public key ipk is not well formed).
- **Zero-Knowledge.** For any label $\ell \in \{0,1\}^*$, when the CRS is generated using $(\mathsf{icrs}, i\mathcal{T}) \overset{\$}{\leftarrow} \mathsf{iTSetup}^{\ell}(\mathsf{crs})$, for any message $\mathsf{x}^* \in i\mathcal{L}$ 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 $i\mathcal{T}$. More formally, we consider the experiments $\mathsf{Exp}^{\mathtt{iZK\text{-}zk\text{-}}b}$ in Figure 2. The iZK is (statistically) zero-knowledge if the advantage of any adversary $\mathcal{A}$ (not necessarily polynomial-time) for these experiments is negligible.

We defined our security notion with a "composable" security flavor, as Groth and Sahai in [GS08]: 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 verified by our constructions and often easier to use.

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

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

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

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

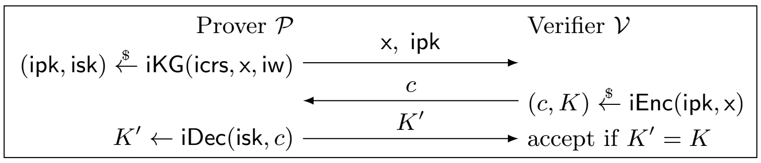

Prover $\mathcal{P}$
Verifier $\mathcal{V}$
$(\mathsf{ipk}, \mathsf{isk}) \xleftarrow{\$} \mathsf{iKG}(\mathsf{icrs}, \mathsf{x}, \mathsf{iw})$
x, ipk
c
$(c, K) \xleftarrow{\$} \mathsf{iEnc}(\mathsf{ipk}, \mathsf{x})$
$K' \leftarrow \mathsf{iDec}(\mathsf{isk}, c)$
K'
accept if $K' = K$

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

– **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 experiments $\mathsf{Exp}^{\mathtt{iZK\text{-}ss\text{-}}b}$ 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 $(\mathsf{ipk}, \mathsf{itk}) \xleftarrow{\$} \mathsf{iTKG}(\mathsf{icrs}, i\mathcal{T}, \mathsf{x})$, 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 add it to L', computes $K \leftarrow \mathsf{iTDec}^\ell(\mathsf{icrs}, \mathsf{itk}, c)$, and outputs K.

  The iZK is (statistically) zero-knowledge if the advantage of any adversary $\mathcal{A}$ (not necessarily polynomial-time) for these experiments is negligible.

Remark 1. An iZK for some language $i\mathscr{L}$ directly leads to a 3-round zero-knowledge arguments for $i\mathscr{L}$. The construction is depicted in Fig. 3 and the proof is provided in Appendix D.4. If the iZK is additionally simulation-sound, the resulting zero-knowledge argument is also simulation-sound.

Remark 2. For the sake of completeness, in Appendix E, 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 [BBC+13] 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:

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

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., $b \in \{0, 1\}$.

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

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

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 above) 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

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

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 [BBC+13] 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:

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

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:

- iSetup(crs) and iTSetup(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 $\mathsf{hk} \xleftarrow{\$} \mathsf{HashKG}(\mathsf{crs}, \mathsf{x})$ 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].

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

[5] The construction can be trivially extended to DLin, or any MDDH assumption [EHK+13] though.

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 Γ 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 $\mathsf{icrs} = (g', h', u' = g'^{r'}, v' = h'^{s'}) \in \mathbb{G}^4$, 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', v') is not a DDH tuple;
- $r' = s'$ a random scalar in $\mathbb{Z}_p$, for the trapdoor CRS generated by $\mathsf{iTSetup}$, with $\mathsf{i}\mathscr{T} = r'$ the trapdoor, so that (g', h', u', v') 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 [ABP14] 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 $\hat{\boldsymbol{C}}_t := (g'^{-1}, 1, 1, 1, 1, 1, 1)$ and $\Gamma_t(C_t) \in \mathbb{G}^{(k+3)\times(n+3)}$ as

$$\Gamma_t(C_t) := \left(\begin{array}{c|cc|c} \multicolumn{3}{c|}{1} & \Gamma(C) \\ \hline g' & 1 & 1 & \hat{\boldsymbol{C}} = \theta(C) \\ \hline 1 & g' & h' & 1 \ \dots \ 1 \\ \hline g' & u' & e' & 1 \ \dots \ 1 \end{array}\right) = \left(\begin{array}{c|cc|cccc} 1 & 1 & 1 & g & h & 1 & 1 \\ 1 & 1 & 1 & 1 & g & u & e/g \\ 1 & 1 & 1 & 1 & 1 & g & h \\ \hline g' & 1 & 1 & u & e & 1 & 1 \\ \hline 1 & g' & h' & 1 & 1 & 1 & 1 \\ \hline g' & u' & e' & 1 & 1 & 1 & 1 \end{array}\right) \qquad (2)$$

Let us show the language corresponding to Γ_t and $\hat{\boldsymbol{C}}_t$ is indeed $\mathscr{L}_t$: Due to the first column of Γ_t and the first element of $\hat{\boldsymbol{C}}_t$, if $\hat{\boldsymbol{C}}_t$ is a linear combination of rows of Γ_t with coefficients $\boldsymbol{\lambda_t}$ (i.e., $\hat{\boldsymbol{C}}_t = \boldsymbol{\lambda_t} \bullet \Gamma_t$), 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.

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

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

  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': $\mathsf{proj}H = \boldsymbol{\lambda'_t} \bullet \mathsf{hp}$ with $\boldsymbol{\lambda'_t} = (0, 0, 0, 0, r', -1)$.

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 $\boldsymbol{\lambda'_t} \bullet \mathsf{hp}$ look random and independent of the real hash value!

[6] Contrary to [ABP14] however, our matrix Γ_t depends on the words C_t, which is why we get this more efficient construction.

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 [GS08] 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 [CS02]. 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 [BBC+13], 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 10). We define an extended language using the generic framework:

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

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

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:

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

so that:

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

```
iSetup(crs)
    $(g', h') \xleftarrow{\$} \mathbb{G}^{*2}$
    $(r', s') \xleftarrow{\$} \mathbb{Z}_p^2 \setminus \{(a, a) \mid a \in \mathbb{Z}_p\}$
    $(u', v') \leftarrow (g'^{r'}, h'^{s'}) \in \mathbb{G}^2$
    $\mathsf{icrs} \leftarrow (u', v')$
    return icrs
```

```
iTSetup(crs)
    $(g', h') \xleftarrow{\$} \mathbb{G}^{*2}$
    $r' \xleftarrow{\$} \mathbb{Z}_p$
    $(u', v') \leftarrow (g'^{r'}, h'^{r'}) \in \mathbb{G}^2$
    $\mathsf{icrs} \leftarrow (u', v')$; $\mathsf{i}\mathcal{T} \leftarrow r'$
    return $(\mathsf{icrs}, \mathsf{i}\mathcal{T})$
```

```
iKG(icrs, x, 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 $(\mathsf{ipk}, \mathsf{isk})$
```

```
iTKG(icrs, x, iT)
    $\mathsf{tk} \xleftarrow{\$} \mathbb{Z}_p^{2k+6}$
    $\mathsf{ipk} := \mathsf{tp} \leftarrow \Gamma_t(\mathsf{x})^\intercal \bullet \mathsf{tk} \in \mathbb{G}^{2n+6}$
    $\mathsf{itk} := (\mathsf{x}, \mathsf{tk}, \mathsf{i}\mathcal{T})$
    return $(\mathsf{ipk}, \mathsf{itk})$
```

```
iEnc(icrs, ipk, x)
    $\mathsf{tp} \leftarrow \mathsf{ipk}$; $\mathsf{hk} \xleftarrow{\$} \mathbb{Z}_p^{n+3}$; $\zeta \xleftarrow{\$} \mathbb{Z}_p$
    $\mathsf{hp} \leftarrow \Gamma_t(\mathsf{x}) \bullet \mathsf{hk} \in \mathbb{Z}_p^{2k+6}$
    $\mathsf{tproj}H \leftarrow \mathsf{hk}^\intercal \bullet \mathsf{tp} \in \mathbb{Z}_p$
    $H \leftarrow \theta_t(\mathsf{x}) \bullet \mathsf{hk} \in \mathbb{Z}_p$
    $K \leftarrow H \cdot \mathsf{tproj}H \in \mathbb{G}$
    $c := (\zeta, \mathsf{hp})$
    return $(K, c)$
```

```
iDec(icrs, 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 $K := \mathsf{proj}H / \mathsf{t}H \in \mathbb{G}$
```

```
iTDec(icrs, 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 $K := \mathsf{trap}H / \mathsf{t}H \in \mathbb{G}$
```

Fig. 4: Construction of iZK

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 (see Appendix C.2 for details). While the previous construction would be useful for security in the UC framework [Can01], 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 are given in Appendix C.1.

3.4 SSiZK Construction

Our SSiZK construction is similar to our iZK construction, except that, in addition both iSetup and iTSetup adds the CRS icrs, a tuple $(v_{k,i})_{i=0,\dots,2\mathfrak{K}}^{k=1,2}$ 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 [Wat05] $\mathcal{W}_k : \{0,1\}^{2\mathfrak{K}} \to \mathbb{G}$, as $\mathcal{W}_k(m) = v_{k,0} \prod_{i=1}^{2\mathfrak{K}} v_{k,i}^{m_i}$, for any bitstring $m = m_1 \| \dots \| m_{2\mathfrak{K}} \in \{0,1\}^{2\mathfrak{K}}$. 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:

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

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 $\hat{\boldsymbol{C}}_t = (g^{-1}, 1, \dots, 1, g^{-\zeta}, 1, \dots, 1)$ (it is the same except for the number of 1's). Due to lack of space, the full matrix is depicted in Appendix D.2, 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 $\mathsf{i}\mathscr{L}$ 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 [HK12] 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 $\mathsf{IP} = \sum_{i=1}^{\ell} x_i y_i \in \{0, \ldots, \ell\}$, 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 Appendix B. 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 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 $(\hat{u}, \hat{v})$ 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 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 as 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 Appendix D.5 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 Appendix A, 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 [GS08] or Σ-protocols. In this section, we sum up our

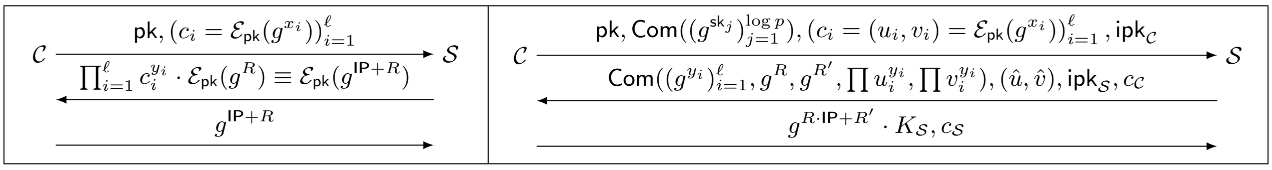

$\mathcal{C}$
$\mathsf{pk}, (c_i = \mathcal{E}_{\mathsf{pk}}(g^{x_i}))_{i=1}^{\ell}$
$\mathcal{S}$
$\prod_{i=1}^{\ell} c_i^{y_i} \cdot \mathcal{E}_{\mathsf{pk}}(g^R) \equiv \mathcal{E}_{\mathsf{pk}}(g^{\mathsf{IP}+R})$
$g^{\mathsf{IP}+R}$
$\mathcal{C}$
$\mathsf{pk}, \mathsf{Com}((g^{\mathsf{sk}_j})_{j=1}^{\log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

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

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. Details are given in Appendix A. 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 (see Appendix A).

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 [Ber06]	no pairings	1	✗
[BGM+10]	≈ 8	≈ 3	≈ 6

Acknowledgments

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

References

ABB+13. M. Abdalla, F. Benhamouda, O. Blazy, C. Chevalier, and D. Pointcheval. SPHF-friendly non-interactive commitments. In *ASIACRYPT 2013, Part I*, *LNCS* 8269, pages 214–234. Springer, December 2013. (Page 3.)

ABP14. M. Abdalla, F. Benhamouda, and D. Pointcheval. Disjunctions for hash proof systems: New constructions and applications. Cryptology ePrint Archive, Report 2014/483, 2014. `http://eprint.iacr.org/2014/483`. (Pages 4, 10, and 31.)

AIR01. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT 2001*, *LNCS* 2045, pages 119–135. Springer, May 2001. (Page 5.)

BBC+13. F. Benhamouda, O. Blazy, C. Chevalier, D. Pointcheval, and D. Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. In *CRYPTO 2013, Part I*, *LNCS* 8042, pages 449–475. Springer, August 2013. (Pages 3, 4, 7, 8, 11, 24, 31, and 32.)

BBP04. M. Bellare, A. Boldyreva, and A. Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *EUROCRYPT 2004*, *LNCS* 3027, pages 171–188. Springer, May 2004. (Page 2.)

Ber06. D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *PKC 2006*, *LNCS* 3958, pages 207–228. Springer, April 2006. (Pages 1, 2, 14, and 16.)

BFI+10. O. Blazy, G. Fuchsbauer, M. Izabachène, A. Jambert, H. Sibert, and D. Vergnaud. Batch groth-sahai. Cryptology ePrint Archive, Report 2010/040, 2010. `http://eprint.iacr.org/2010/040`. (Page 19.)

BGM+10. J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal Ate pairing over Barreto-Naehrig curves. In *PAIRING 2010*, *LNCS* 6487, pages 21–39. Springer, December 2010. (Pages 14 and 16.)

BPV12. O. Blazy, D. Pointcheval, and D. Vergnaud. Round-optimal privacy-preserving protocols with smooth projective hash functions. In *TCC 2012*, *LNCS* 7194, pages 94–111. Springer, March 2012. (Page 3.)

BPW12. D. Bernhard, O. Pereira, and B. Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In *ASIACRYPT 2012*, *LNCS* 7658, pages 626–643. Springer, December 2012. (Page 1.)

BR93. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73. ACM Press, November 1993. (Page 1.)

BR09. M. Bellare and T. Ristenpart. Simulation without the artificial abort: Simplified proof and improved concrete security for Waters' IBE scheme. In *EUROCRYPT 2009*, *LNCS* 5479, pages 407–424. Springer, April 2009. (Page 26.)

Bro13. J. Brodkin. Satellite internet faster than advertised, but latency still awful, February 2013. http://arstechnica.com/information-technology/2013/02/satellite-internet-faster-than-advertised-but-latency. (Page 1.)

Can00. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000. (Page 28.)

Can01. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. (Page 12.)

CGH04. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004. (Page 2.)

CHK$^+$05. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT 2005*, *LNCS* 3494, pages 404–421. Springer, May 2005. (Page 29.)

CS02. R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT 2002*, *LNCS* 2332, pages 45–64. Springer, April / May 2002. (Pages 3, 4, and 11.)

DDN91. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography (extended abstract). In *23rd ACM STOC*, pages 542–552. ACM Press, May 1991. (Page 1.)

EHK$^+$13. A. Escala, G. Herold, E. Kiltz, C. Ràfols, and J. Villar. An algebraic framework for Diffie-Hellman assumptions. In *CRYPTO 2013, Part II*, *LNCS* 8043, pages 129–147. Springer, August 2013. (Page 9.)

FLS90. U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990. (Page 3.)

FS87. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, *LNCS* 263, pages 186–194. Springer, August 1987. (Page 1.)

GGSW13. S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. In *45th ACM STOC*, pages 467–476. ACM Press, June 2013. (Page 5.)

GIKM98. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *30th ACM STOC*, pages 151–160. ACM Press, May 1998. (Page 5.)

GL03. R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. In *EUROCRYPT 2003*, *LNCS* 2656, pages 524–543. Springer, May 2003. http://eprint.iacr.org/2003/032.ps.gz. (Page 3.)

GL06. R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. *ACM Transactions on Information and System Security*, 9(2):181–234, 2006. (Page 3.)

GMR89. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. (Page 1.)

GMW87a. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, pages 218–229. ACM Press, May 1987. (Pages 1 and 4.)

GMW87b. O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO'86*, *LNCS* 263, pages 171–185. Springer, August 1987. (Pages 1, 4, and 20.)

GMY03. J. A. Garay, P. D. MacKenzie, and K. Yang. Strengthening zero-knowledge protocols using signatures. In *EUROCRYPT 2003*, *LNCS* 2656, pages 177–194. Springer, May 2003. (Page 19.)

Gol04. O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, New York, NY, USA, 2004. (Page 20.)

GS08. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT 2008*, *LNCS* 4965, pages 415–432. Springer, April 2008. (Pages 1, 2, 4, 5, 6, 11, 13, 16, and 32.)

HJ12. D. Hofheinz and T. Jager. Tightly secure signatures and public-key encryption. In *CRYPTO 2012*, *LNCS* 7417, pages 590–607. Springer, August 2012. (Page 18.)

HK12. D. Hofheinz and E. Kiltz. Programmable hash functions and their applications. *Journal of Cryptology*, 25(3):484–527, July 2012. (Page 13.)

HKE13. Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO 2013, Part II*, *LNCS* 8043, pages 18–35. Springer, August 2013. (Page 4.)

II. E. II. ebats. http://bench.cr.yp.to/results-dh.html. (Page 1.)

IKLP06. Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank. Black-box constructions for secure computation. In *38th ACM STOC*, pages 99–108. ACM Press, May 2006. (Page 4.)

Lin13. Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO 2013, Part II*, *LNCS* 8043, pages 1–17. Springer, August 2013. (Page 4.)

LP07. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT 2007*, *LNCS* 4515, pages 52–78. Springer, May 2007. (Page 4.)

LP11. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC 2011*, *LNCS* 6597, pages 329–346. Springer, March 2011. (Page 4.)

LPJY14. B. Libert, T. Peters, M. Joye, and M. Yung. Non-malleability from malleability: Simulation-sound quasi-adaptive NIZK proofs and CCA2-secure encryption from homomorphic signatures. In *EUROCRYPT 2014*, *LNCS* 8441, pages 514–532. Springer, May 2014. (Page 26.)

Mau09. U. M. Maurer. Unifying zero-knowledge proofs of knowledge. In *AFRICACRYPT 09*, *LNCS* 5580, pages 272–286. Springer, June 2009. (Pages 19 and 20.)

MTVY11. T. Malkin, I. Teranishi, Y. Vahlis, and M. Yung. Signatures resilient to continual leakage on memory and computation. In *TCC 2011*, *LNCS* 6597, pages 89–106. Springer, March 2011. (Page 4.)

NY90. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990. (Page 1.)

Sch90. C.-P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, *LNCS* 435, pages 239–252. Springer, August 1990. (Pages 1 and 4.)

sS11. a. shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In *EUROCRYPT 2011*, *LNCS* 6632, pages 386–405. Springer, May 2011. (Page 4.)

sS13. a. shelat and C.-H. Shen. Fast two-party secure computation with minimal assumptions. In *ACM CCS 13*, pages 523–534. ACM Press, November 2013. (Page 4.)

Wat05. B. R. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT 2005*, *LNCS* 3494, pages 114–127. Springer, May 2005. (Pages 4, 12, and 26.)

Yao86. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. (Page 1.)

A Details on the Inner Product Protocols

We will now provide a detailed analysis of the performances of our UC-secure protocol to compute the inner product. Next, we compare the performances to the performances of a similar protocol whose security is based on the Groth-Sahai methodology [GS08] to illustrate the fact that, in applications where pairings are not fundamentally required for the protocol (meaning, the semi-honest version of the protocol can be done without pairings), being able to avoid them allows us to provide way more efficient solutions. We also provide the performances of a protocol based on the Schnorr proofs (Σ-proofs), which implies more interactivity.

A.1 Intuition on the Efficiency Improvements

First, let us provide an intuition of the reasons why we can expect some efficiency improvement over round-efficient protocols in the malicious setting based on NIZK.

Avoiding Pairings Saves Computations. Pairing are an expensive operation; on the best known curves such as [BGM+10], computing a pairing is roughly three time slower than computing an exponentiation. Moreover, not every elliptic curve has a pairing, and it turns out that the most efficient curves, such as [Ber06], have indeed no pairings. In the best curves without pairings, exponentiations in $\mathbb{G}$ are roughly three times faster than exponentiations in $\mathbb{G}_1$ in the best curves with pairings, and even six times faster than exponentiations in $\mathbb{G}_2$.

iZK Can be Efficiently Batched. iZK are somewhat "batch-friendly": batch techniques, which reduce computation and communication, can always be used with an iZK without requiring more interactions. To batch a proof in NIZK-based protocols a seed is needed, so the prover has to first commit to his values, then he receives the seed and computes a short NIZK from it, that he sends back. This adds two rounds compared to the classical one-flow protocol in which the prover directly sends commitments plus a NIZK. But with iZKs, things are different: the prover sends commitments and an ipk, and the verifier replies with the next flow encrypted with ipk. It turns out that the prover and the verifier can agree on a batched version of the proof before even knowing the seed, so the prover can compute ipk without knowing the seed, and the verifier can just send the seed together with the masked second flow. Consequently, we can apply batch techniques to iZK-based protocols to reduce the communication without adding interactivity.

The Conversion of iZK into SSiZK is Efficient. We presented in Section 3 a generic construction of SSiZK from iZK. It is worth mentioning that this construct is *efficient* as it only adds a small constant number of group elements to the original iZK. Conversely, turning NIZK into simulation-sound NIZK comes at huge cost, a linear blow-up of the size of the proof. As soon as strong security requirements are considered, such as security in the UC framework, simulation-sound zero-knowledge proofs become, in the general case, unavoidable.

A.2 Setup

Let us provide some details about the iZK proofs which ensure the security of the inner product protocol described in Section 4. We work in a cyclic group $\mathbb{G}$ of prime order p, where the DDH assumption holds. We denote by λ the bit length of p, and by g a generator of the group. We also set the Cramer-Shoup public key to $\left(g_1, g_2, a, b, (h_i)_{i=1}^{\lambda+\ell+2}\right)$, together with a universal hash function $\mathsf{H}(\cdot)$. Since we apply the randomness-reuse technique for the Cramer-Shoup encryption, we need as many group elements h_i as the maximal size of the vector we will encrypt. The value $\lambda + \ell + 2$ is a clear upper-bound. The group description and this key (to be used for the commitment) are in the CRS.

Committing to the Secret Key. As described in Section 4, the client has to commit to her secret key; such a commitment adds $O(\lambda)$ to the communication complexity of the protocol. However, the same requirement holds for any secure variant of the inner product protocol (based on the Groth-Sahai methodology or based on Σ-proofs), so, for the sake of simplicity, we omit this commitment (and the proof that it is indeed the secret key) in the protocols we are going to compare. The reason is that we focus on the setting $\ell \gg \lambda$ (for example, in the biometric setting, we can have $\lambda = 128$ while $\ell \approx 2000$) so this $O(\lambda)$ will not affect the overall comparison, even though the constants can differ from one protocol to the other.

A.3 Inner Product Protocol with iZK

Equations for the Language of the First Flow ($\mathsf{i}\mathscr{L}_\mathcal{C}$). These equations ensure that all the encrypted values are bits. the ϵ_i denote random values (used to batch the equations) which do not appear in the matrix of the SPHF associated to the iZK, so they will be picked by the server once he received the ciphertexts. $\mathsf{i}\mathscr{L}_\mathcal{C}$ is the language of words $(u_i^{\epsilon_i}, v_i^{\epsilon_i})_{i\leq\ell}$ such that there exists $((r_i, x_i)_{i\leq\ell}, \mu)$ satisfying:

1. for $i = 1$ to ℓ, $u_i^{\epsilon_i} = g^{\epsilon_i r_i}$ and $v_i^{\epsilon_i} = h^{\epsilon_i r_i} g^{\epsilon_i x_i}$
2. $1 = (\prod u_i^{\epsilon_i x_i}) \cdot g^{-\mu}$ and $1 = (\prod (v_i/g)^{\epsilon_i x_i}) \cdot h^{-\mu}$

The $2\ell + 2$ equations involve $2\ell + 1$ witnesses, $((\epsilon_i r_i)_{i\leq\ell}, (\epsilon_i x_i)_{i\leq\ell}, \mu)$. The witness μ corresponds to $\sum \epsilon_i r_i x_i$. Omitting the constants, this lead to an iZK with public key $\mathsf{ipk}_\mathcal{C}$ of size 4ℓ and ciphertext $c_\mathcal{C}$ of size 4ℓ.

Equations for the Language of the Second Flow ($\mathsf{i}\mathscr{L}_\mathcal{S}$). Let $(d_1, d_2, (e_i)_{i\leq\ell+4}, f)$ denote the Cramer-Shoup commitments of the values $((g^{y_i})_{i\leq\ell}, g^R, g^{R'}, \prod u_i^{y_i}, \prod v_i^{y_i})$, and let $(\hat{u}, \hat{v})$ denote the encryption of $R \cdot \mathsf{IP} + R'$. These equations ensure that the first ℓ committed values are bits, that the two last committed values are $\prod u_i^{y_i}$ and $\prod v_i^{y_i}$ and that $(\hat{u}, \hat{v})$ is a randomized encryption of the inner product additively and multiplicatively randomized by two committed values. The values are committed using randomness reuse techniques, which makes the commitment four times smaller but prevents us from batching our equations as we did in the first flow. $\mathsf{i}\mathscr{L}_\mathcal{S}$ is the language of words $(d_1, d_2, (e_i)_{i\leq\ell+4}, f, \hat{u}, \hat{v})$ such that there exists $((y_i)_{i\leq\ell}, (\mu_i)_{i\leq\ell+1}, r', R, R', \rho)$ satisfying:

1. $d_1 = g_1^{r'}, d_2 = g_2^{r'}, f = (ab^\xi)^{r'}$
2. for $i = 1$ to ℓ, $e_i = h_i^{r'} g^{y_i}, 1 = d_1^{y_i} g^{-\mu_i}$ and $1 = (e_i/g)^{y_i} h_i^{-\mu_i}$
3. $e_{\ell+1} = h_{\ell+1}^{r'} g^R, e_{\ell+2} = h_{\ell+2}^{r'} g^{R'}$ ans $1 = d_1^R g^{-\mu_{\ell+1}}$
4. $e_{\ell+3} = h_{\ell+3}^{r'} \prod u_i^{y_i}, e_{\ell+4} = h_{\ell+4}^{r'} \prod v_i^{y_i}$

5. $\hat{u} = g^{\rho} e_{\ell+3}^{R} h_{\ell+3}^{-\mu_{\ell+1}}, \hat{v} = h^{\rho} e_{\ell+4}^{R} h_{\ell+4}^{-\mu_{\ell+1}} g^{R'}$

The $3\ell + 10$ equations involve $2\ell + 5$ witnesses, $((y_i)_{i\leq\ell}, (\mu_i)_{i\leq\ell+1}, r', R, R', \rho)$. The witnesses $(\mu_i)_{i\leq\ell}$ correspond to the $r'y_i$'s and $\mu_{\ell+1}$ corresponds to $r'R$. ρ is the randomness used to randomize the ciphertext $(\hat{u}, \hat{v})$. Omitting the constants, the corresponding SSiZK has a public key $\mathsf{ipk}_{\mathcal{S}}$ of size 4ℓ and ciphertext $c_{\mathcal{S}}$ of size 6ℓ.

Communication Complexity. omitting the constants, the total communication complexity of the protocol, counting the ciphertexts, the commitments, the iZK and the SSiZK, is $2\ell+4\ell+4\ell+\ell+4\ell+6\ell = 21\ell$.

Computational Complexity. Exponentiations are required to compute the ciphertexts, the commitments, and elements of the iZK involved in the two iZKs: $(\mathsf{hp}, \mathsf{tp}, H, \mathsf{t}H, \mathsf{proj}H, \mathsf{tproj}H)$. Recall that as $(x_i, y_i)_{i\leq\ell}$ are bits, exponentiations with these values are free.

– First iZK: $4 \times 5\ell$ (for hp and tp), plus $2 \times 2\ell$ (for H and $\mathsf{tproj}H$), plus $2 \times 2\ell$ (for $\mathsf{t}H$ and $\mathsf{proj}H$), plus $2 \times 2\ell$ (for the ElGamal ciphertexts). Hence 30ℓ exponentiations in total.
– Second iZK: $4 \times 6\ell$ (for hp and tp), plus $2 \times 2\ell$ (for H and $\mathsf{tproj}H$), plus $2 \times 4\ell$ (for $\mathsf{t}H$ and $\mathsf{proj}H$), plus $2 \times \ell$ (for the commitments). Hence 37ℓ exponentiations in total.

Omitting the constants, the execution of the whole protocol requires 67ℓ exponentiations.

A.4 Inner Product Protocol with Groth-Sahai NIZKs

Unlike our iZK-based protocol, we do not intend to fully construct a UC-secure protocol for the inner product with the Groth-Sahai methodology, but rather to provide a lower bound on the complexity of such a protocol, which is enough to assess our claim that iZKs provide consistent efficiency improvement over the Groth-Sahai methodology to design UC-secure protocols whose semi-honest version does not originally involve pairings. Notice that we can apply batch techniques to reduce drastically the number of equations needed for the NIZK of the first flow, as the client cannot gain knowledge from the second flow by cheating (IP is randomized by R and R'), but the same cannot be done for the second flow because the client cannot send the decrypted value without being sure that the server was honest.

Simulation-Soundness for Groth-Sahai NIZKs. The inner product protocol involves quadratic equations and pairing product equations. While very efficient (quasi-adaptive) simulation-sound NIZKs have been designed for linear equations, to our knowledge, the best simulation-sound NIZKs for quadratic equations and pairing product equations are those of [HJ12]. However, the conversion of a NIZK into a simulation-sound NIZK with this method incurs a huge additive overhead (because of the signature) and a linear blow-up of the size of the NIZK. As the conversion of NIZK into simulation-sound NIZK involves precise computations and optimizations, we have *not* attempted to evaluate it in this section; as a consequence, all the estimations are (loose) *lower bounds* on the real complexity of a Groth-Sahai-based UC-secure inner product protocol.

Communication Complexity. To prove that all the committed values are bits, which is a quadratic equation, the x_i's have to be committed over $\mathbb{G}_1$ and $\mathbb{G}_2$, and the randomness of the ElGamal ciphertexts $(r_i)_{i\leq\ell}$ has to be committed over $\mathbb{G}_2$. These commitments and the ciphertexts represent in total 4ℓ group elements over $\mathbb{G}_1$ and 4ℓ group elements over $\mathbb{G}_2$. However, all the equations (checking that the ElGamal ciphertexts are well-formed, checking that values committed over $\mathbb{G}_1$ and $\mathbb{G}_2$ are indeed the same, checking that all the x_i are bits) can be batched. For the second flow, the server has to send commitments of the y_i's over $\mathbb{G}_1$ and $\mathbb{G}_2$, together with encryptions of the y_i's (required for the simulatability, but randomness reuse can be applied here to reduce linearly the number of group elements) and commitments over $\mathbb{G}_2$ of the randomness of the encryptions of the y_i's. Moreover, proving that the y_i are bits involves ℓ quadratic equations, which represents 2ℓ elements over $\mathbb{G}_1$ and 2ℓ elements over $\mathbb{G}_2$. As we explained, we cannot batch those equations without adding two rounds to the protocol.

The proof that the ciphertexts do indeed encrypt the committed values costs ℓ group elements over $\mathbb{G}_1$ and proving that values committed over $\mathbb{G}_1$ and $\mathbb{G}_2$ are indeed the same costs at least ℓ elements over $\mathbb{G}_1$. Thus, the second flow contains at least 7ℓ group elements over $\mathbb{G}_1$ and 6ℓ group elements over $\mathbb{G}_2$.

Total. the communication complexity of the whole execution of a UC-secure inner product protocol using the Groth-Sahai methodology is lower bounded by 11ℓ group elements over $\mathbb{G}_1$ and 10ℓ group elements over $\mathbb{G}_2$. $\mathbb{G}_2$ being approximately twice as big as $\mathbb{G}_1$ with usual settings, this represents roughly 31ℓ elements over $\mathbb{G}_1$, which is 50% more than the iZK-based protocol.

Pairings and Exponentiations. Counting the number of exponentiations of Groth-Sahai proofs is quite involved, as this number is quadratic $O(\ell^2)$ in the general case, but linear in nearly every specific application, if the correct optimizations are used. Instead of counting the exponentiations, we focus on a loose lower bound by counting only the exponentiations required to compute ciphertexts and commitments, without even considering the computations required for the construction and the verification of the proofs. this leads to a lower bound of 28ℓ exponentiations over $\mathbb{G}_2$ ans 6ℓ exponentiations over $\mathbb{G}_1$. Moreover, several paper have lowered the number of pairing needed to verify the proofs; even if we consider that the verification of all the proofs can be batched into a single verification of a pairing-product equation, using the optimizations of [BFI+10], at least 4ℓ pairings are required for the first flow. For the second flow, which cannot be batched, verification (using [BFI+10]) of one pairing-product equation, two multi-scalar multiplication equations and one quadratic equation is lower bounded by $(4+2+2+2)\ell = 10\ell$ pairings. The overall number of pairings is thus lower-bounded by 14ℓ. As we can choose more efficient curves, with fast exponentiations, by avoiding the need of pairings, even these very loose values represent considerably more computations that the exponentiations required by the iZK-based protocol.

A.5 Inner Product Protocol with Schnorr Σ-Protocols

Let us now provide an estimation of the cost of an UC-secure protocol for the inner product relying on Σ-Protocols (i.e., protocols with a three-move structure, namely (*commitments*, *challenge*, *response*)). There are two ways of designing such a protocol:

1. One can rely on the OR trick to prove, for each ciphertext (u, v), that either (u, v) or (u, gv^{-1}) is an encryption of 0 (a DDH tuple).
2. Alternatively, one can commit to $(x_i^2)_{i\le\ell}$, prove that the commitments contains the square of the encrypted values (using a Chaum-Pedersen proof of same discrete logarithm with different bases), and then batch all the proofs by proving a statement of the form $\sum_{i=1}^{\ell} \lambda_i(x_i - x_i^2) = 0$, for a random tuple of values $(\lambda_i)_{i\le\ell}$ chosen by the verifier after the prover has committed.

We will focus on the second technique for our estimation; both techniques seem roughly equivalent in terms of communication and computation. The commitment scheme used in this protocol is the Pedersen commitment scheme, which can be seen as the second part of an ElGamal ciphertext: $c(m; r) = h^r g^m$. The reader might refer to [Mau09] to get an intuition of the cost of the different proofs we are going to construct, as all our proofs can be seen as proving the knowledge of a preimage of a group homomorphism, which fits into the framework of [Mau09]. Moreover, all those proofs can be turned into simulation-sound ZK proofs at a small, constant additive cost, using the generic transformation of [GMY03]. The protocol goes as follow: (we omit the constants when we provide the number of elements exchanged)

Protocol.

1. The client sends ℓ ElGamal ciphertexts $(u_i, v_i)_{i\le\ell}$ and ℓ commitments $(w_i)_{i\le\ell}$ of the squares of the encrypted values. He also generates 3ℓ randomness for the proof, hash them using a collision-resistant hash function, and commits to this value.
2. The server replies with a challenge c, ℓ ElGamal ciphertexts of his own values (required for the simulatability) plus the randomness (R, R') (with his key), ℓ commitments of the squares of his values and an encryption (with the client key) of $(R \cdot \mathsf{IP} + R')$.

3. The client sends a proof, which contains 3ℓ scalars and 3ℓ openings of the randomness whose hash value he committed to in the first flow. he also sends a challenge c'.
4. The server checks that the openings are correct, and if they are, that the proofs hold, i.e.that the values where indeed bits and that $(\lambda_i)_{i \leq \ell}$, the values λ_i being computed from the challenge c with a pseudo-random generator. Then, he sends himself a similar proof, ensuring his values are bits ($3\ell + 3\ell$ elements), plus a proof that the randomized scalar product was correctly computed (2ℓ elements).
5. If the openings and the proofs are correct, the client sends the decrypted randomized inner product to the server.

For details on how Σ-protocols can be built for statements such as "I know openings of commitments such that one of them opens to the product of the two other committed values", the reader might refer to [Mau09]. We enhance the security of the original Σ-protocols by adding commitments to the randomness and revealing the openings after receiving the challenge; such enhanced protocols can be proven secure against malicious verifiers, and so are truly zero-knowledge.

Efficiency. The communication complexity can be easily counted from our description of the protocol: $(2 + 1 + 2 + 1 + 3 + 3 + 3 + 3 + 2)\ell = 20\ell$. The computational complexity, counted as a number of exponentiations and omitting constant values and other operations, is 38ℓ:

– $2\ell + \ell$ for the ciphertexts and Pedersen commitments of the first flow.
– $3\ell + 3\ell$ for the random ciphertexts and Pedersen commitments hashed and committed in the first flow.
– $2\ell + \ell$ for the ciphertexts and Pedersen commitments of the second flow.
– $3\ell + 3\ell$ for the random ciphertexts and Pedersen commitments hashed and committed in the second flow.
– $3\ell + 2\ell$ to check the opening of the random ciphertexts and Pedersen commitments hashed and committed in the first flow.
– $3\ell + 2\ell$ to check the proofs (3ℓ for the commitments of squares of encrypted values, 2ℓ for the batched proof of bit values)
– $3\ell + 2\ell$ to check the opening of the random ciphertexts and Pedersen commitments hashed and committed in the second flow.
– $3\ell + 2\ell$ to check the proofs (3ℓ for the commitments of squares of encrypted values, 2ℓ for the batched proof of bit values).
– 2ℓ to check the proof that the inner product was correctly computed.

B Semi-Honest to Malicious Transformation

In the seminal work [GMW87b], Goldreich, Micali and Wigderson have proven that there exists a compiler which, given any two-party semi-honest interactive protocol, outputs an "equivalent protocol" for the malicious model. This compiler (which we call GMW compiler) is formally described in [Gol04]. It is divided in three phases: the Input-Commitment Phase, where the players commit to their own inputs; the Coin-Generation Phase, where the players run an augmented coin-tossing protocol to generate unbiased random tapes while providing commitments on them for later validity proofs; and the Protocol Emulation Phase, where zero-knowledge proofs are used to ensure semi-honest behavior of all the players, from the committed inputs, the committed random tapes and the flows. This last phase is the one on which we focus in this section.

Indeed, while NIZK could be used to prove correct generation of the flows, they would either be quite inefficient (with general NIZK constructions) or require strong settings and assumptions (assumptions in bilinear groups for Groth-Sahai NIZK). On the other hand, interactive zero-knowledge proofs imply a blow-up in the interactivity of the protocol.

We present another compiler (see Figure 6) which is divided in four phases: there are still the Input-Commitment Phase and the Coin-Generation Phase, which end up with commitments of the inputs and of the unbiased random tapes of the two players, as in the GMW compiler. Note that if inputs should belong in a non-trivial language, validity of the commitments has to be proven as in the next phase. These are constant-round phases, which are then followed by the $\mathsf{Protocol\ Emulation\ Phase}$: each flow x from the initial protocol is combined with an iZK, and so with a public key ipk, so that the other player can mask all the subsequent flows with K (or derivative masks) encapsulated in c. More precisely, from the ephemeral key K, we write $k^{(i)}$ for $\mathrm{PRG}^{(i)}(K)$, and each flow is masked by all the previous keys, and so we use the next block from the PRG for any new mask. Hence, as soon as one player tries to cheat, all the subsequent flows sent by the other player will be masked by a random value. Eventually, a $\mathsf{Verification\ Phase}$ provides an explicit validity check: the two players have to prove they were able to extract all the ephemeral keys, which guarantees their semi-honest behavior during the whole protocol.

Proof Sketch. For the security proof, we first assume we are dealing with a deterministic function: on private inputs x and y, the first player receives $f(x, y)$ and the second receives $g(x, y)$. For the sake of simplicity, we also make the assumption that the semi-honest protocol provides execution traces with formats (size and number of flows) that are independent of the inputs. Eventually, we make use of extractable commitments.

We are thus given a simulator $\mathcal{S}im$ for the semi-honest protocol $\mathcal{P}$. And we describe a simulator $\mathcal{S}im'$ for the compiled protocol $\mathcal{P}'$: If both players are honest, $\mathcal{S}im'$ simply runs the simulator $\mathcal{S}im$ to generate all the basic flows, and generates all the iZK proofs as well as the verification flows, but using random keys K for deriving the masks. If one player is malicious, $\mathcal{S}im'$ first extracts its inputs and random coins from the extractable commitment, sends the inputs to the ideal functionality to learn the outcome and provides it to $\mathcal{S}im$ to generate the basic flows of the honest player. This time, valid iZK proofs for the flows of the honest player have to be generated since the malicious player will be able to check them, and $\mathcal{S}im'$ has to be able to immediately detect dishonest behavior of the malicious player in order to replace all the subsequent flows by random flows: the trapdoor for the iZK, in the CRS, allows $\mathcal{S}im'$ to extract the ephemeral key even without a witness, and then to get back the plaintext sent by the malicious player; from the inputs and the random tape of the malicious player, as well as the previous flows already exchanged, $\mathcal{S}im'$ can anticipate and check the flow that should have been generated with a semi-honest behavior. As soon as a cheating attempt is detected, in the real world, the subsequent masks would become random looking to the malicious player, $\mathcal{S}im'$ can thus safely send random flows (the masked parts).

C More Efficient iZK Constructions

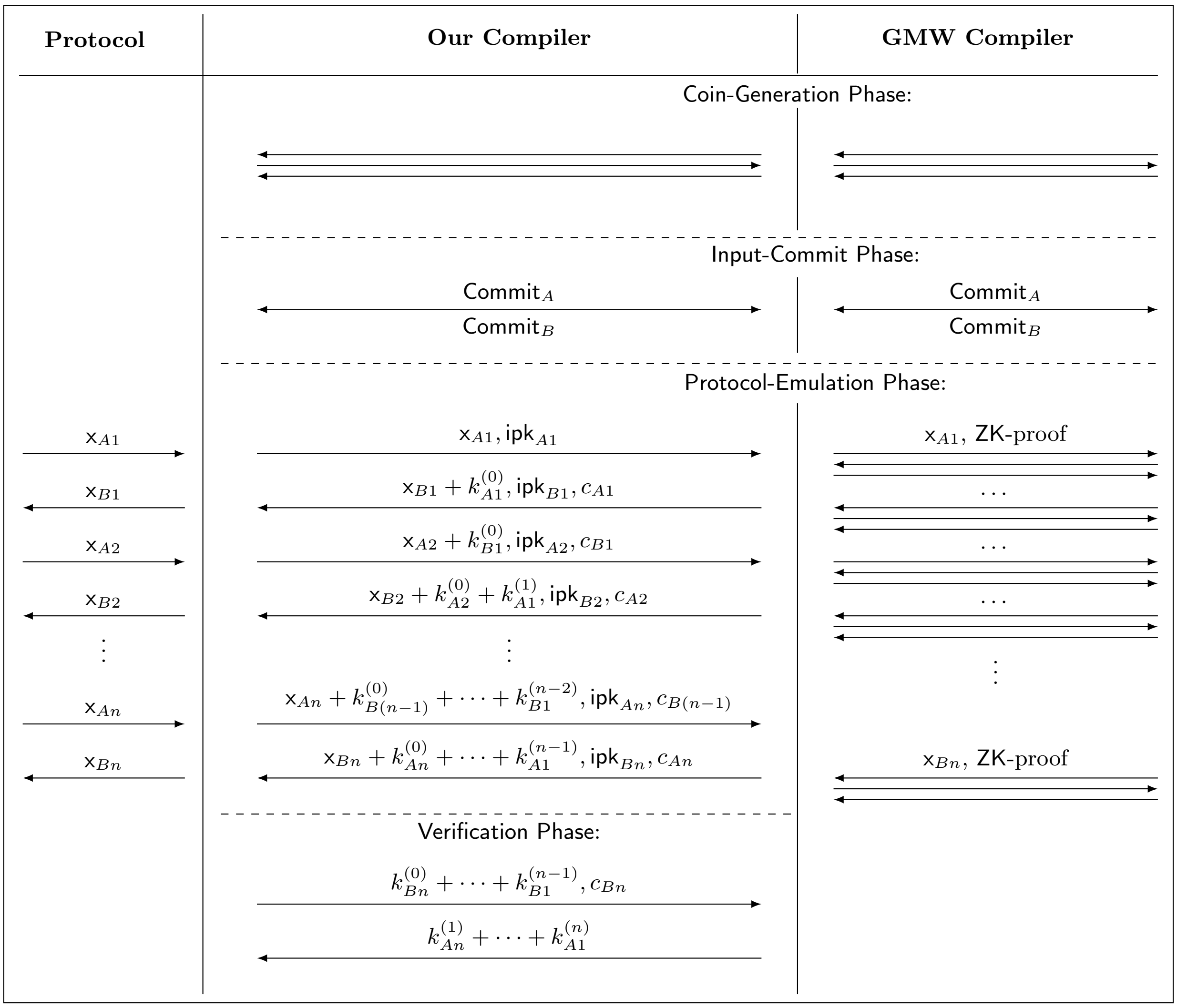
In this section, we describe several ways to get slightly more efficient constructions of iZK at the cost of some (very reasonable) additional requirements.

C.1 Reducing the Size of the Ciphertext Using Entropy Extractors

In the generic framework constructed in Section 3, the ciphertext c of the iZK contains a random integer ζ, which is fundamental to ensure the 2-universality property. However, the actual requirement on ζ is quite simple: we want to ensure that the adversary will not be able to guess it before we send it. If the adversary was able to guess ζ, then he could have sent a $\mathsf{tproj}H$ corresponding to a linear combination of the lines of the matrix, and then $\mathsf{tproj}H$ would contain additional information about the secret key, breaking the zero-knowledge property of the iZK. To ensure that the adversary will not guess the ζ in advance, it is not necessary to send the ζ among with the other elements of the ciphertext c, as it already contains a lot of entropy: one can add the description of an entropy extractor Ext in the CRS, and the value ζ will be directly computed as $\zeta = \mathsf{Ext}(\mathsf{hp})$. This saves one element in c.

C.2 More Efficient Construction with Word-Dependent CRS

In section 3, we have seen how to add a trapdoor in a SPHF to ensure the validity of the projection key. In many cases, it is possible to add the trapdoor in a slightly more efficient way, if we accept to use word-dependent CRS. (the trapdoor CRS only works for one word $\mathsf{x}^* = (u^*, e^*)$ chosen before the CRS



Protocol
Our Compiler
GMW Compiler
Coin-Generation Phase:
Input-Commit Phase:
Commit$_A$
Commit$_B$
Commit$_A$
Commit$_B$
Protocol-Emulation Phase:
x_{A1}
x_{B1}
x_{A2}
x_{B2}
$\vdots$
x_{An}
x_{Bn}
$\mathsf{x}_{A1}, \mathsf{ipk}_{A1}$
$\mathsf{x}_{B1} + k_{A1}^{(0)}, \mathsf{ipk}_{B1}, c_{A1}$
$\mathsf{x}_{A2} + k_{B1}^{(0)}, \mathsf{ipk}_{A2}, c_{B1}$
$\mathsf{x}_{B2} + k_{A2}^{(0)} + k_{A1}^{(1)}, \mathsf{ipk}_{B2}, c_{A2}$
$\vdots$
$\mathsf{x}_{An} + k_{B(n-1)}^{(0)} + \cdots + k_{B1}^{(n-2)}, \mathsf{ipk}_{An}, c_{B(n-1)}$
$\mathsf{x}_{Bn} + k_{An}^{(0)} + \cdots + k_{A1}^{(n-1)}, \mathsf{ipk}_{Bn}, c_{An}$
x_{A1}, ZK-proof
$\cdots$
$\cdots$
$\cdots$
$\vdots$
x_{Bn}, ZK-proof
Verification Phase:
$k_{Bn}^{(0)} + \cdots + k_{B1}^{(n-1)}, c_{Bn}$
$k_{An}^{(1)} + \cdots + k_{A1}^{(n)}$

Fig. 6: Semi-honest to malicious compilers

```
$\mathsf{Exp}^{\mathtt{iZK\text{-}zk\text{-}}b}(\mathcal{A}, \mathsf{crs}, \mathfrak{R})$
    $(\mathsf{x}^*, w, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(\mathsf{crs})$                    $\triangleright$ only for word-dependent CRS
    $\mathsf{x}^* \leftarrow \perp$                                          $\triangleright$ only for re-usable CRS
    $(\mathsf{icrs}, \mathsf{i}\mathcal{T}) \xleftarrow{\$} \mathsf{iTSetup}(\mathsf{crs}, \mathsf{x}^*)$
    $(\ell, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(\mathsf{crs})$                             $\triangleright$ only for word-dependent CRS
    $(\ell, \mathsf{x}^*, w, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(\mathsf{st}, \mathsf{icrs}, \mathsf{i}\mathcal{T})$               $\triangleright$ only for re-usable CRS
    if $\mathcal{R}(\mathsf{x}^*, w, \mathsf{st}) = 0$ then return $0$
    if $b = 0$ then
        $(\mathsf{ipk}, \mathsf{isk}) \xleftarrow{\$} \mathsf{iKG}^\ell(\mathsf{icrs}, \mathsf{x}^*, w)$
    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 $\mathcal{A}(\mathsf{st}, K)$
```

Fig. 7: Experiments $\mathsf{Exp}^{\mathtt{iZK\text{-}zk\text{-}}b}$ for zero-knowledge of iZK

is generated). Instead of adding three columns and three rows to the matrix Γ (to obtain the matrix Γ_t), it may be possible to only add one row. The second part of the construction ensuring the validity of the projection keys hp_t remains the same.

For example, in Example 6, the CRS can contain a row $R = (R_1, R_2, R_3, R_4)$ which is $(u^{*s}, e^{*s}, 1, 1)$ in the trapdoor mode for x^*, or $(g^s, h^s, 1, 1)$ in the normal mode (with s a random scalar in $\mathbb{Z}_p^*$). In the trapdoor mode, s is the trapdoor for x^*. The DDH (or the semantic security of ElGamal) ensures that the two setups are indistinguishable. We then have (if we omit the 2-universal trick at the end):

$$\hat{\boldsymbol{C}}_t = \hat{\boldsymbol{C}} = (u, e, 1, 1)$$

$$\Gamma_t = \left(\frac{\Gamma}{R}\right) = \begin{pmatrix} g & h & 1 & 1 \\ 1 & g & u & e/g \\ 1 & 1 & g & h \\ R_1 & R_2 & R_3 & R_4 \end{pmatrix}.$$

In normal mode, the last row R is s times the first row of Γ_t, and so the new element in the projection key, $\mathsf{hp}_{t,4} = R \bullet \mathsf{hk}$ gives no more information than the first element $\mathsf{hp}_{t,1} = \mathsf{hp}_1$ (from an information theoretic point of view). That is why the smoothness does still hold in normal mode.

In trapdoor mode, we remark that $\mathsf{hp}_{t,4} = R \bullet \mathsf{hk} = (u^{*\mathsf{hk}_1} e^{*\mathsf{hk}_2})^s$. This is exactly the hash value of x^* raised to the power of s (if hp is valid). So knowing the trapdoor s and $\mathsf{hp}_{t,4}$ enables to compute the hash value of C^*.

Formal Construction of iZK with Word-Dependent CRS. We suppose to have two setup algorithms:

- $\mathsf{iSetup}(\mathsf{crs})$ generates a row vector $R \in \mathbb{G}^{1 \times n}$ which is linearly depend of the rows of any matrix Γ for any C (we recall that Γ may depend on C). Then it returns $\mathsf{icrs} = (\mathsf{crs}, R)$.
- $\mathsf{iTSetup}(\mathsf{crs}, \mathsf{x}^*)$ generates a row vector $R \in \mathbb{G}^{1 \times n}$ and a trapdoor a row vector $\boldsymbol{\lambda}^* \in \mathbb{Z}_p^{k+1}$ so that:

$$\boldsymbol{\lambda}^* \bullet \left(\frac{\Gamma}{R}\right) = \hat{\boldsymbol{C}}.$$

In other word $\boldsymbol{\lambda}^*$ is a witness for the language defined by $\left(\frac{\Gamma}{R}\right)$ and θ. Then it returns $\mathsf{icrs} = (\mathsf{crs}, R)$ and $\mathsf{i}\mathcal{T} = \boldsymbol{\lambda}^*$.

Then we do the same construction as in Section 3.3, except we use the following matrices $\Gamma'_t(\mathsf{x})$, $\hat{\boldsymbol{C}}_t$, $\boldsymbol{\lambda_t}(\zeta, \mathsf{iw})$, and $\boldsymbol{\lambda_t}(\zeta, \mathsf{i}\mathcal{T})$:

$$\hat{\boldsymbol{C}}_t = (\hat{\boldsymbol{C}}, \zeta \bullet \hat{\boldsymbol{C}}) \qquad\qquad \Gamma'_t = \left(\frac{\Gamma}{R}\right)$$

$$\boldsymbol{\lambda_t}(\zeta, \mathsf{iw}) = (\boldsymbol{\lambda}, 0, \zeta \bullet \boldsymbol{\lambda}, 0) \qquad\qquad \boldsymbol{\lambda_t}(\zeta, \mathsf{i}\mathcal{T}) = (\boldsymbol{\lambda}^*, \zeta \bullet \boldsymbol{\lambda}^*).$$

If the two setup iSetup and $\mathsf{iTSetup}$ are indeed indistinguishable, we prove the construction to be secure in Appendix D.3. The proof is very similar to the one for the generic construction in Appendix D.1.

When it is usable, this construction is slightly more efficient than the generic one with re-usable CRS, since the resulting matrix Γ_t has 4 less columns and 4 less rows.

D Proofs

D.1 Proof of the iZK Construction of Section 3

Correctness. Straightforward.

Setup Indistinguishability. The only difference between iSetup and $\mathsf{iTSetup}$ is that in the former (g', h', u', e') is a random tuple, while in the later (g', h', u', e') is a DDH tuple. Hence the setup indistinguishability holds under plain DDH in $\mathbb{G}$.

Soundness. Let us consider a CRS $\mathsf{icrs} = (\mathsf{crs}, g', h', u', e')$ generated by $\mathsf{iSetup}(\mathsf{crs})$. We need to show that, for any $C = \mathsf{x} \notin \mathscr{L} = \mathsf{i}\mathscr{L}$ (i.e., such that $\hat{\boldsymbol{C}}$ is linearly independent of rows of Γ) and any $\mathsf{iZK} = \mathsf{tp} \in \mathbb{G}^{2n+6}$, the distribution of $K = H \cdot \mathsf{tproj}H$ is statistically close to uniform over $\mathbb{G}$, even given $c = (\zeta, \mathsf{hp})$. For that purpose, we will prove something stronger: with overwhelming probability over ζ, the distribution of H is uniform even given $\mathsf{hp}, \zeta, \mathsf{tproj}H$.

The key idea for the demonstration is to apply the same argument than in the demonstration of the smoothness of the generic construction of [BBC+13] on a well chosen matrix Γ_s such that the projection key for this matrix is exactly $\mathsf{hp}_s = (\mathsf{hp}, \mathsf{tproj}H)$, but the language is not changed (with overwhelming probability over ζ). As $\mathsf{hp} = \Gamma_t \bullet \mathsf{hk}$ and $\mathsf{tproj}H = \mathsf{tp} \bullet \mathsf{hk}$, we can set

$$\Gamma_s = \left(\frac{\Gamma_t}{\mathsf{tp}}\right), \qquad\qquad \mathsf{hp}_s = (\mathsf{hp}, \mathsf{tproj}H) = \Gamma_s \bullet \mathsf{hk}.$$

Let us prove that with overwhelming probability over ζ, $\hat{\boldsymbol{C}}_t$ is linearly independent of rows of Γ_s. This will prove that with overwhelming probability, $H = \hat{\boldsymbol{C}}_t \bullet \mathsf{hk}$ looks uniformly random in $\mathbb{G}$, even given $\mathsf{hp}_s = (\mathsf{hp}, \mathsf{tproj}H)$.

More precisely, let us prove that there is at most one value $\zeta \in \mathbb{Z}_p$ such that $\hat{\boldsymbol{C}}_t = (g'^{-1}, 1, \dots, 1, g'^{-\zeta}, 1, \dots, 1)$ is linearly dependent of rows of Γ_s. Let us suppose by contradiction, that there exists $\zeta \neq \zeta'$, $\boldsymbol{\lambda}_s = (\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \mu)$, and $\boldsymbol{\lambda'}_s = (\boldsymbol{\lambda'_1}, \boldsymbol{\lambda'_2}, \mu')$ such that $(g'^{-1}, 1, \dots, 1, g'^{-\zeta}, 1, \dots, 1) = \boldsymbol{\lambda}_s \bullet \Gamma_s$, and $(g'^{-1}, 1, \dots, 1, g'^{-\zeta}, 1, \dots, 1) = \boldsymbol{\lambda'}_s \bullet \Gamma_s$. This implies

$$(g'^{-1}, 1, \dots, 1) = \boldsymbol{\lambda_1} \bullet \Gamma' + \mu \bullet \mathsf{tp}_1 \tag{3a}$$
$$(g'^{-\zeta}, 1, \dots, 1) = \boldsymbol{\lambda_2} \bullet \Gamma' + \mu \bullet \mathsf{tp}_2 \tag{3b}$$
$$(g'^{-1}, 1, \dots, 1) = \boldsymbol{\lambda'_1} \bullet \Gamma' + \mu' \bullet \mathsf{tp}_1 \tag{3c}$$
$$(g'^{-\zeta'}, 1, \dots, 1) = \boldsymbol{\lambda'_2} \bullet \Gamma' + \mu' \bullet \mathsf{tp}_2 \tag{3d}$$

with $1 = g^0 \in \mathbb{G}$ and $\mathsf{tp}_1, \mathsf{tp}_2 \in \mathbb{G}^{n+3}$ such that $\mathsf{tp} = (\mathsf{tp}_1, \mathsf{tp}_2)$, and

$$\Gamma' = \left(\begin{array}{c|cc|c} \multicolumn{3}{c|}{1} & \Gamma \\ \hline g' & 1 & 1 & \hat{\boldsymbol{C}} \\ \hline 1 & g' & h' & 1 \;\dots\; 1 \\ \hline g' & u' & e' & 1 \;\dots\; 1 \end{array}\right) \in \mathbb{G}^{(k+3)\times(n+3)}.$$

Then, we get:

$$(g'^{-\mu'+\mu}, 1, \ldots, 1) = (\mu' \bullet \boldsymbol{\lambda_1} - \mu \bullet \boldsymbol{\lambda'_1}) \bullet \Gamma' \qquad (\mu' \bullet (3a) - \mu \bullet (3c))$$
$$(g'^{-\zeta\mu'+\zeta'\mu}, 1, \ldots, 1) = (\mu' \bullet \boldsymbol{\lambda_2} - \mu \bullet \boldsymbol{\lambda'_2}) \bullet \Gamma' \qquad (\mu' \bullet (3b) - \mu \bullet (3d))$$

If $\mu' = \mu$, $-\zeta\mu' + \zeta'\mu = \zeta' - \zeta \neq 0$, otherwise $-\mu' + \mu \neq 0$. By dividing the first equation by $-\mu' + \mu$ in the latter case, or the second equation by $-\zeta\mu' + \zeta'\mu$ in the former case, we get that there exists some vector $\boldsymbol{\lambda'} \in \mathbb{Z}_p^{k+3}$ such that:

$$(g'^{-1}, 1, \ldots, 1) = \boldsymbol{\lambda'} \bullet \Gamma'.$$

But since (g', h', u', e) is not a DDH tuple, λ' has to be of the form $(\star, \ldots, \star, 1, 0, 0)$, which means that $\hat{\boldsymbol{C}}$ is a linear combination of rows of Γ, i.e., $C \in \mathscr{L}$, which is not the case.

Therefore, our construction is sound (and the two distributions we consider in the definition in Section 2.3.

Zero-Knowledge. Let $\mathsf{x}^* \in \mathsf{i}\mathscr{L} = \mathscr{L}$ be a word with witness iw^*. For the zero-knowledge property, we (the challenger playing the role of the prover) generates a public key $\mathsf{ipk} = \mathsf{tp}$, where tp is a projection key, associated to a random hashing key tk, for the language of valid hp's. Then, the adversary (playing the role of the verifier) sends a ciphertext $c(\zeta, \mathsf{hp})$. There are two cases:

– either there exists $\mathsf{hk} \in \mathbb{Z}_p^{2n+6}$ such that $\mathsf{hp} = \Gamma_t \bullet \mathsf{hk}$. In this case, we have

$$\begin{aligned}\mathsf{proj}H &:= \boldsymbol{\lambda_t}(\zeta, \mathsf{iw}^*) \bullet \mathsf{hp} = \boldsymbol{\lambda_t}(\zeta, \mathsf{iw}^*) \bullet \Gamma_t \bullet \mathsf{hk} = \hat{\boldsymbol{C}}_t \bullet \mathsf{hk} \\ &= \boldsymbol{\lambda_t}(\zeta, \mathsf{i}\mathcal{T}) \bullet \Gamma_t \bullet \mathsf{hk} = \boldsymbol{\lambda_t}(\zeta, \mathsf{i}\mathcal{T}) \bullet \mathsf{hp} := \mathsf{trap}H\end{aligned}$$

  (this property actually can be seen as coming from the correctness of the SPHF with projection key hp);

– or, there does not exist $\mathsf{hk} \in \mathbb{Z}_p^{2n+6}$ such that $\mathsf{hp} = \Gamma_t \bullet \mathsf{hk}$. In this case, hp is not valid and $\mathsf{t}H = \Gamma_t^{\mathsf{T}} \bullet \mathsf{tk}$ (with $\mathsf{tk} \in \mathbb{Z}_p^{2k+6}$) looks uniformly random for the adversary (before he sees $\mathsf{proj}H \cdot \mathsf{t}H$ or $\mathsf{trap}H \cdot \mathsf{t}H$ in the game), since the only information he sees about tk is $\mathsf{tp} = \Gamma_t^{\mathsf{T}} \bullet \mathsf{tk}$, but hp is linearly independent of rows of Γ_t^{T}. This property on $\mathsf{t}H$ can actually be seen as the smoothness property of the SPHF with projection key tp. Then $\mathsf{proj}H \cdot \mathsf{t}H$ and $\mathsf{trap}H \cdot \mathsf{t}H$ looks both uniformly random to the adversary, and cannot be distinguished.

Therefore, our construction is perfect zero-knowledge.

D.2 Details and Proof of the SSiZK Construction of Section 3.4

Details. Let us first write down the complete matrices $\hat{\boldsymbol{C}}_t$ and $\Gamma_t(\mathsf{x})$:

$$\theta_t(\mathsf{x}, \zeta) = \hat{\boldsymbol{C}}_t = (g'^{-1}, 1, \ldots, 1, g'^{-\zeta}, 1, \ldots, 1) \qquad \in \mathbb{G}^{1 \times (2n+10)}$$

$$\Gamma'_t(\ell, \mathsf{x}) = \left(\begin{array}{c|cc|ccc|cc} \multicolumn{3}{c|}{1} & \multicolumn{3}{c|}{\Gamma(\mathsf{x})} & \multicolumn{2}{c}{1} \\ \hline g' & 1 & 1 & \multicolumn{3}{c|}{\hat{\boldsymbol{C}}} & 1 & 1 \\ \hline\hline 1 & g' & h' & 1 & \ldots & 1 & 1 & 1 \\ g' & u' & e' & 1 & \ldots & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & \ldots & 1 & g' & h' \\ 1 & 1 & 1 & 1 & \ldots & 1 & u'' & e'' \\ \hline g' & 1 & 1 & 1 & \ldots & 1 & g' & 1 \end{array}\right) \qquad \in \mathbb{G}^{(k+6) \times (n+5)}$$

$$\Gamma_t(\ell, \mathsf{x}) = \left(\begin{array}{c|c} \Gamma'_t(\ell, \mathsf{x}) & \mathbf{1} \\ \hline \mathbf{1} & \Gamma'_t(\ell, \mathsf{x}) \end{array}\right) \qquad \in \mathbb{G}^{(2k+12) \times (2n+10)},$$

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

with $u'' = \mathcal{W}_1(\mathcal{H}(\ell, \mathsf{x}))$ and $e'' = \mathcal{W}_2(\mathcal{H}(\ell, \mathsf{x}))$.

Proof.

Correctness. Straightforward.

Setup Indistinguishability. The only elements added to the CRS $(v_{1,i}, v_{2,i})_i$ have exactly the same distribution when generated by iSetup and $\mathsf{iTSetup}$. So it is equivalent to the setup indistinguishability of our iZK construction (see Appendix D.1), and is implied by the DDH assumption.

Zero-Knowledge. The proof is exactly the same as for our iZK construction (see Appendix D.1).

Soundness. Both iSetup and $\mathsf{iTSetup}$ output a CRS icrs, such that $(g', h', v_{1,i}, v_{2,i})$ is a DDH tuple, and so is (g', h', u'', v''). From the definition of $\hat{\boldsymbol{C}}_t$ and $\varGamma_t$, a word $(\mathsf{x}, g', h', u', v', u'', v'')$ is in the extended language corresponding to $\hat{\boldsymbol{C}}_t$ and $\varGamma_t$ if and only if $\mathsf{x} \in \mathsf{i}\mathscr{L}$, or (g', h', u', v') is a DDH tuple, or $(g', 1)$ is in the subspace generated by (g', h') and (u'', v''). But the latter subspace is exactly the subspace generated by (g', h') (as (g', h', u'', v'') is a DDH tuple). Hence, $(g', 1)$ is never in that subspace (as g' and h' are supposed to be generators), and the last case of the disjunction is never satisfied.

Therefore, the extended language is actually the same as for our iZK construction, and the soundness can be proved in the same way as in Appendix D.1.

Simulation-Soundness. Let us now prove the simulation-soundness by exhibiting a sequence of indistinguishable games. An overview of the proof is given in Section 3.4.

We consider an adversary $\mathcal{A}$ against the simulation soundness. In each game $\mathbf{G}_i$, we start by picking a random bit b, run some experiment, and output some bit b'. We denote by Adv_i the advantage of the adversary in the game $\mathbf{G}_i$:

$$\mathsf{Adv}_i = 2 \cdot \Pr[b' = b] - 1.$$

Finally, we write negl any negligible quantity in $\mathfrak{K}$.

We recall that we suppose 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 $\mathsf{i}\mathscr{L}$ or not. This makes the check $\mathsf{x}^* \in \mathsf{i}\mathscr{L}^*$.

Game $\mathbf{G}_0$: In this first game, we pick a random bit b, run the experiment $\mathsf{Exp}^{\mathtt{iZK\text{-}ss\text{-}}b}(\mathcal{A}, \mathsf{crs}, \mathfrak{K})$, and outputs the bit b' (output by the experiment). The advantage Adv_0 is exactly the advantage of the adversary $\mathcal{A}$ in the simulation soundness experiments.

Game $\mathbf{G}_1$: In this game, instead of picking DDH tuples $(g', h', v_{1,i}, v_{2,i})$ in $\mathsf{iTSetup}$, we pick $v_{1,i}$ and $v_{2,i}$ uniformly at random in $\mathbb{G}$. Under the DDH assumption, $\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \text{negl}$.

Game $\mathbf{G}_2$: Similarly to the proof in [LPJY14], in this game, we pick $g'', h'' \xleftarrow{\$} \mathbb{G}$, and set, for $i = 0, \ldots, 2\mathfrak{K}$:

$$r'_i \xleftarrow{\$} \mathbb{Z}_p \qquad r''_i \xleftarrow{\$} \mathbb{Z}_p \tag{4}$$

$$v_{1,i} \leftarrow g'^{r'_i} \cdot g''^{r''_i} \cdot g'^{\rho'_i} \qquad v_{2,i} \leftarrow h'^{r'_i} \cdot h''^{r''_i}, \tag{5}$$

with $\rho'_0 = \mu\zeta' - \rho_0$, $\rho'_i = -\rho_i$ (for $i = 1, \ldots, 2\mathfrak{K}$), $\mu \xleftarrow{\$} \{0, \ldots, 2\mathfrak{K}\}$, $r'_i, r''_i \xleftarrow{\$} \mathbb{Z}_p$, $\rho_i \xleftarrow{\$} \{0, \ldots, \zeta'\}$, for $i = 0, \ldots, 2\mathfrak{K}$, with $\zeta' = 2(q+1)$ and q the number of simulated proofs (i.e., queries (ℓ, x) to oracle $\mathcal{O}$). This game is perfectly indistinguishable from the previous one, as the distribution of the $v_{k,i}$'s is exactly the same: $\mathsf{Adv}_1 = \mathsf{Adv}_2$.

Game $\mathbf{G}_3$: In this game, we abort if for some query (ℓ, x) to $\mathcal{O}$, $\rho'_0 + \sum_{i=i}^{2\mathfrak{K}} m_i \rho'_i = 0$, with $m = m_1 \| \ldots \| m_{2\mathfrak{K}} = \mathcal{H}(\ell, \mathsf{x}) \in \{0,1\}^{2\mathfrak{K}}$; or if for $m^* = m^*_1 \| \ldots \| m^*_{2\mathfrak{K}} = \mathcal{H}(\ell^*, \mathsf{x}^*) \in \{0,1\}^{2\mathfrak{K}}$, $\rho'_0 + \sum_{i=1}^{2\mathfrak{K}} m^*_i \rho'_i \neq 0$. Using the same analysis as in [Wat05, BR09, LPJY14]: $\mathsf{Adv}_2^2 / (27(q+1)(2\mathfrak{K}+1)) \leq \mathsf{Adv}_3$.

Game $\mathbf{G}_4$: In this game, we choose g'', h'' so that (g', h', g'', h'') is a random DDH tuple (instead of a random tuple as before). Under the DDH assumption, $\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \text{negl}$.

Game $\mathbf{G}_5$: In this game, we set, for $i = 0, \dots, 2\mathfrak{K}$:

$$r'_i \xleftarrow{\$} \mathbb{Z}_p \qquad\qquad v_{1,i} \leftarrow g'^{r'_i} \cdot g'^{\rho'_i} \qquad\qquad v_{2,i} \leftarrow h'^{r'_i}, \tag{6}$$

with ρ_i defined as in $\mathbf{G}_2$. This game is perfectly indistinguishable from the previous one, as the distribution of the $v_{k,i}$'s is exactly the same: $\mathsf{Adv}_4 = \mathsf{Adv}_5$.

Game $\mathbf{G}_6$: In this game, for any query (ℓ, x) to $\mathcal{O}$, we generate $\mathsf{ipk} = \mathsf{tp}$ as usual, but for a subsequent query $(\mathsf{ipk} = \mathsf{tp}, c = (\zeta, \mathsf{hp}))$ to $\mathcal{O}$, we compute $\mathsf{trap}H$ (in iTDec) as $\mathsf{trap}H = \boldsymbol{\lambda}' \bullet \mathsf{hp}$ instead of $\mathsf{trap}H = \boldsymbol{\lambda_t}(\zeta, \mathsf{i}\mathcal{T}) \bullet \mathsf{hp}$, where

$$\boldsymbol{\lambda}' = \left(0, \dots, 0, -\frac{r'_0 + \sum_{i=1}^{2\mathfrak{K}} r'_i}{\alpha}, \frac{1}{\alpha}, -1, 0, \dots, 0, -\zeta\frac{r'_0 + \sum_{i=1}^{2\mathfrak{K}} r'_i}{\alpha}, \frac{\zeta}{\alpha}, -\zeta\right),$$

and

$$m = \mathcal{H}(\ell, \mathsf{x}) \qquad\qquad \alpha = \rho'_0 + \sum_{i=1}^{n} m_i \rho'_i.$$

This vector $\boldsymbol{\lambda}'$ is well defined as $\alpha \neq 0$ from the abort condition in $\mathbf{G}_3$. Furthermore:

$$\begin{aligned}
&\begin{pmatrix} \frac{r'_0+\sum_{i=1}^{2\mathfrak{K}} r'_i}{\alpha} & \frac{1}{\alpha} \end{pmatrix} \bullet \begin{pmatrix} g' & h' \\ \mathcal{W}_1(m) = v_{1,0}\prod_{i=1}^{2\mathfrak{K}} v_{1,i}^{m_i} & \mathcal{W}_2(m) = v_{2,0}\prod_{2=1}^{2\mathfrak{K}} v_{1,i}^{m_i} \end{pmatrix} \\
&= \begin{pmatrix} g'^{(-r'_0-\sum_{i=1}^{2\mathfrak{K}} m_i r'_i)/\alpha} g'^{(r'_0+\sum_{i=1}^{2\mathfrak{K}} m_i r'_i+\rho'_0+\sum_{i=1}^{2\mathfrak{K}} \rho'_i)/\alpha} \\ h'^{(-r'_0-\sum_{i=1}^{2\mathfrak{K}} m_i r'_i)/\alpha} h'^{(r'_0+\sum_{i=1}^{2\mathfrak{K}} m_i r'_i)/\alpha} \end{pmatrix}^{\mathsf{T}} \\
&= \begin{pmatrix} g'^{(\rho'_0+\sum_{i=1}^{2\mathfrak{K}} \rho'_i))/\alpha} & h'^{0} \end{pmatrix} = \begin{pmatrix} g' & 1 \end{pmatrix}
\end{aligned}$$

so that

$$\boldsymbol{\lambda}' \bullet \mathit{\Gamma}_t = \hat{\boldsymbol{C}}_t.$$

Finally, a proof similar as the one for the zero-knowledge property of our iZK construction (see Appendix D.1) shows that $\mathsf{Adv}_5 \leq \mathsf{Adv}_6 + \text{negl}$.

Game $\mathbf{G}_7$: In this game, we generate the CRS using iSetup (i.e., (g', h', u', e') is now a random tuple instead of a DDH tuple). This is possible as $\mathsf{i}\mathcal{T}$ was not used in the previous game. Under the DDH assumption, $\mathsf{Adv}_6 \leq \mathsf{Adv}_7 + \text{negl}$.

In this last game, we remark that $\hat{\boldsymbol{C}}_t^*$ (corresponding to the challenge ℓ^*, x^*) is linearly independent of rows of $\mathit{\Gamma}_t(\ell^*, \mathsf{x}^*)$, as $\mathsf{x}^* \notin \mathsf{i}\mathscr{L}$, (g', h', u', e') is not a DDH tuple, and $(g', h', \mathcal{W}_1(\ell^*, \mathsf{x}^*), \mathcal{W}_2(\ell^*, \mathsf{x}^*))$ is a DDH tuple. Then, similarly as in the soundness proof above, we get that $\mathsf{Adv}_7 = \text{negl}$ (statistically).

D.3 Proof of iZK Construction for Word-Dependent CRS of Appendix C.2

Correctness. Straightforward.

Setup Indistinguishability. This is an assumption for this construction.

Soundness. The proof is very similar to the one in Appendix D.1. As usual, we write $C = \mathsf{x}$ and $\mathscr{L} = \mathsf{i}\mathscr{L}$. We just need to prove that there is at most one value $\zeta \in \mathbb{Z}_p$ such that $\hat{\boldsymbol{C}}_t = (\hat{\boldsymbol{C}}, \zeta \bullet \hat{\boldsymbol{C}})$ is linearly dependent of rows of $\mathit{\Gamma}_s$ when $C \notin \mathscr{L}$, where $\mathit{\Gamma}_s$ is defined by $\mathit{\Gamma}_s = \begin{pmatrix} \mathit{\Gamma}_t \\ \mathsf{tp} \end{pmatrix}$. Let us suppose by contradiction, that there exists $\zeta \neq \zeta'$, $\boldsymbol{\lambda}_s = (\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \mu)$, and $\boldsymbol{\lambda}'_s = (\boldsymbol{\lambda}'_1, \boldsymbol{\lambda}'_2, \mu')$ such that $(\hat{\boldsymbol{C}}, \zeta \bullet \hat{\boldsymbol{C}}) = \boldsymbol{\lambda}_s \bullet \mathit{\Gamma}_s$, and $(\hat{\boldsymbol{C}}, \zeta \bullet \hat{\boldsymbol{C}}) = \boldsymbol{\lambda}'_s \bullet \mathit{\Gamma}_s$. This implies

$$\hat{\boldsymbol{C}} = \boldsymbol{\lambda_1} \bullet \mathit{\Gamma}' + \mu \bullet \mathsf{tp}_1 \tag{7a}$$

$$\zeta \bullet \hat{\boldsymbol{C}} = \boldsymbol{\lambda_2} \bullet \mathit{\Gamma}' + \mu \bullet \mathsf{tp}_2 \tag{7b}$$

$$\hat{\boldsymbol{C}} = \boldsymbol{\lambda}'_1 \bullet \mathit{\Gamma}' + \mu' \bullet \mathsf{tp}_1 \tag{7c}$$

$$\zeta \bullet \hat{\boldsymbol{C}} = \boldsymbol{\lambda}'_2 \bullet \mathit{\Gamma}' + \mu' \bullet \mathsf{tp}_2 \tag{7d}$$

with $\mathsf{tp}_1, \mathsf{tp}_2 \in \mathbb{G}^n$ such that $\mathsf{tp} = (\mathsf{tp}_1, \mathsf{tp}_2)$, and

$$\Gamma' = \begin{pmatrix} \Gamma \\ R \end{pmatrix}.$$

Then, we get:

$$(-\mu' + \mu) \bullet \hat{\boldsymbol{C}} = (\mu' \bullet \boldsymbol{\lambda_1} - \mu \bullet \boldsymbol{\lambda_1'}) \bullet \Gamma' \qquad (\mu' \bullet (7a) - \mu \bullet (7c))$$
$$(-\zeta\mu' + \zeta'\mu) \bullet \hat{\boldsymbol{C}} = (\mu' \bullet \boldsymbol{\lambda_2} - \mu \bullet \boldsymbol{\lambda_2'}) \bullet \Gamma' \qquad (\mu' \bullet (7b) - \mu \bullet (7d))$$

If $\mu' = \mu$, $-\zeta\mu' + \zeta'\mu = \zeta' - \zeta \neq 0$, otherwise $-\mu' + \mu \neq 0$. By dividing the first equation by $-\mu' + \mu$ in the latter case, or the second equation by $-\zeta\mu' + \zeta'\mu$ in the former case, we get that there exists some vector $\boldsymbol{\lambda'} \in \mathbb{Z}_p^{k+1}$ such that:

$$\hat{\boldsymbol{C}} = \boldsymbol{\lambda'} \bullet \Gamma' = \boldsymbol{\lambda'} \bullet \begin{pmatrix} \Gamma \\ R \end{pmatrix}.$$

But since R is linearly dependent of rows of Γ, when the CRS is generated by iSetup, this means that $\hat{\boldsymbol{C}}$ is also a linear combination of rows of Γ, i.e., $C \in \mathscr{L}$, which is not the case. Hence the soundness property holds.

Zero-Knowledge. The proof is almost identical to the one in Appendix D.1.

D.4 Proof of Construction of ZK Arguments from iZK

We formally prove that the construction of ZK from iZK given in Remark 1 is correct. Let us first recall the definition of an **interactive protocol**. Let us consider a language $\mathscr{L}$. A n-round interactive protocol $(\mathcal{P}, \mathcal{V})$ is any pair of randomized algorithms (not necessarily computationally bounded). The interaction between $\mathcal{P}$ and $\mathcal{V}$ on a word $x \in \mathcal{X}$ is depicted in Fig. 8. The verifier can do two other actions: accept or reject the word x at any time. We write $(\mathcal{P}, \mathcal{V})(x) = 1$ if the verifier accepts during the interaction and $(\mathcal{P}, \mathcal{V})(x) = 0$ otherwise.

Now, we recall the definition of a zero-knowledge proof: a zero-knowledge proof on a word $C \in \mathscr{L}$ is an interactive protocol $(\mathcal{P}, \mathcal{V})$ verifying the following properties:

– **Completeness**. $(\mathcal{P}, \mathcal{V})$ is complete, if for any $C \in \mathscr{L}$:

$$\Pr\left[\, (\mathcal{P}, \mathcal{V})(x) \neq 1 \,\right] \leq \operatorname{negl}(\mathfrak{K});$$

– **Soundness**. $(\mathcal{P}, \mathcal{V})$ is sound, if for any $C \in \mathcal{X} \setminus \mathscr{L}$, for any prover $\mathcal{P}'$:

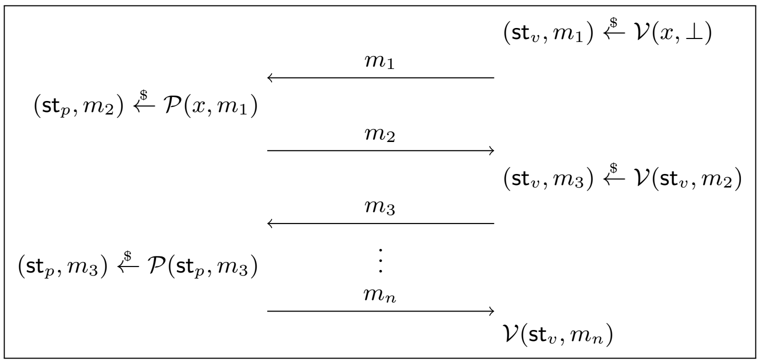
$$\Pr\left[\, (\mathcal{P}', \mathcal{V})(x) = 1 \,\right] \leq \operatorname{negl}(\mathfrak{K});$$

– **Zero-Knowledge**. $(\mathcal{P}, \mathcal{V})$ is zero-knowledge, if for any probabilistic polynomial-time adversary playing the role of the verifier, honestly or not, there exists a polynomial time simulator $\mathcal{S}im$ able to simulate the view of this adversary without using a witness. The view $\langle \mathcal{P}, \mathcal{V} \rangle (x)$ of a verifier is the tuple $(m_1, \ldots, m_n, r)$ of the exchanged messages and its random tape. More formally, $(\mathcal{P}, \mathcal{V})$ is computationally zero-knowledge if, for all $C \in \mathscr{L}$, for any PPT A, the distributions $\langle \mathcal{P}, A \rangle (x)$ and $\mathcal{S}im(x)$ are computationally indistinguishable.

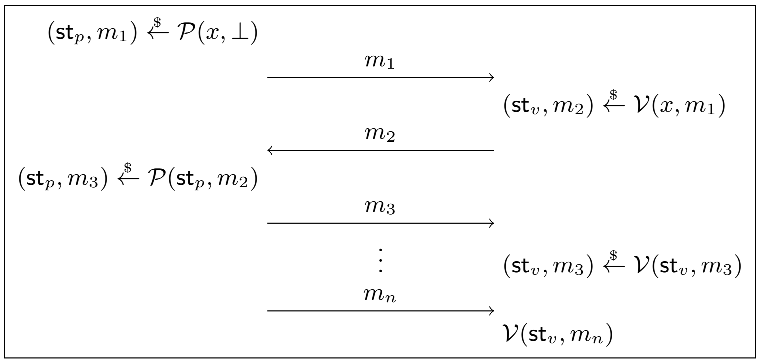
Hence, the completeness and the soundness of the zero-knowledge protocol from iZK directly follows from the completeness and the soundness of the underlying iZK; the zero-knowledge is straightforward too: the existence of a simulator is ensured because a simulator is explicitly given by the underlying iZK. The simulator simply uses the trapdoor instead of the witness, and the proof of perfect simulation directly follows from the zero-knowledge property of the underlying iZK.

D.5 Proof of Security of our Inner Product Protocol in the UC Model

In this section, we prove that (the malicious version of) our scheme in Section 4 is secure in the UC model [Can00], with authenticated channels and static corruptions.


$(\mathsf{st}_v, m_1) \xleftarrow{\$} \mathcal{V}(x, \bot)$
m_1
$(\mathsf{st}_p, m_2) \xleftarrow{\$} \mathcal{P}(x, m_1)$
m_2
$(\mathsf{st}_v, m_3) \xleftarrow{\$} \mathcal{V}(\mathsf{st}_v, m_2)$
m_3
$(\mathsf{st}_p, m_3) \xleftarrow{\$} \mathcal{P}(\mathsf{st}_p, m_3)$
m_n
$\mathcal{V}(\mathsf{st}_v, m_n)$

(a) n is even


$(\mathsf{st}_p, m_1) \xleftarrow{\$} \mathcal{P}(x, \bot)$
m_1
$(\mathsf{st}_v, m_2) \xleftarrow{\$} \mathcal{V}(x, m_1)$
m_2
$(\mathsf{st}_p, m_3) \xleftarrow{\$} \mathcal{P}(\mathsf{st}_p, m_2)$
m_3
$(\mathsf{st}_v, m_3) \xleftarrow{\$} \mathcal{V}(\mathsf{st}_v, m_3)$
m_n
$\mathcal{V}(\mathsf{st}_v, m_n)$

(b) n is odd

Fig. 8: Interactive protocol execution for a word $x \in \mathcal{X}$

Details on the Scheme. Here are some implicit details related to UC for the scheme in Section 4: all flows contains an identifier (1 for the first flow, 2 for the second flow and 3 for the third flow). Every flow not formatted correctly is ignored. Every commitment is supposed to be labeled with an identifier of the commitment (1 for the one of the first flow and 2 for the one of the second flow), the identifier of $\mathcal{C}$ and $\mathcal{S}$, the session and sub-session identifiers sid and ssid. We use the labeled version of the Cramer-Shoup encryption scheme [CHK+05] for that purpose. We recall that this scheme is $\mathsf{IND\text{-}CCA}$ secure.

Ideal Functionality. The ideal functionality is depicted in Fig. 9. Basically, the client $\mathcal{C}$ sends its input $(x_i)_{i=1}^{\ell} \in \{0,1\}^{\ell}$, then the server sends its input $(y_1)_{i=1}^{\ell}$, and finally, when the adversary or simulator $\mathcal{S}im$ specifies it, the server gets back the inner product IP of (x_i) and (y_i). Corruptions of the client or the server are supposed to be static, i.e., before the first message `Client-Send` is sent (for a given session $(\mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S})$). The authentication and the flow identifiers above ensure that if one of the player is honest at the beginning, he remains honest during all the session and the adversary in the real world cannot modify his flow (though he may drop them as usual and attempt a denial-of-service attack).

Proof of Security. We exhibit a sequence of games. The sequence starts from the real game, where the adversary $\mathcal{A}$ interacts with real players and ends with the ideal game, where we have built a simulator $\mathcal{S}im$ that makes the interface between the ideal functionality $\mathcal{F}$ and the adversary $\mathcal{A}$.

Game $\mathbf{G}_0$: This is the real game, where the simulator knows the inputs of all the honest players and honestly play their role (on their behalf).

Game $\mathbf{G}_1$: We first deal with the case when $\mathcal{C}$ *and* $\mathcal{S}$ *are both honest*. In that case, the simulator $\mathcal{S}im$ replaces all commitments and ciphertexts of $\mathcal{C}$ and $\mathcal{S}$ by commitments and ciphertexts of random values. In addition, except if the adversary $\mathcal{A}$ drops some flows, the simulator $\mathcal{S}im$ never abort on behalf of $\mathcal{S}$ and outputs the correct inner product $\mathsf{IP} = \sum_{i=1}^{\ell} x_i \cdot y_i$ he can compute since he still knows the inputs (x_i) of $\mathcal{C}$ and (y_i) of $\mathcal{S}$. $\mathcal{S}im$ also sends to the message `Result-Send` when

The functionality $\mathcal{F}_{\mathsf{IP}}$ is parametrized by a security parameter k. It interacts with an adversary $\mathcal{S}im$ and a set of parties via the following queries:

Upon receiving a query $(\texttt{Client-Send}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S}, (x_i)_{i=1}^{\ell})$ from party $\mathcal{C}$ (client): Ignore the message if $(x_i)_{i=1}^{\ell} \notin \{0,1\}^{\ell}$. Record the tuple $(\mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S}, (x_i))$ and send $(\texttt{Client-Sent}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S})$ to $\mathcal{S}im$. Ignore further $\texttt{Client-Send}$-message with the same $(\mathsf{ssid}, \mathcal{C}, \mathcal{S})$ from $\mathcal{C}$.

Upon receiving a query $(\texttt{Server-Send}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S}, (y_i)_{i=1}^{\ell})$ from party $\mathcal{S}$ (client): Ignore the message if $(y_i)_{i=1}^{\ell} \notin \{0,1\}^{\ell}$. Ignore the message if $(\mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S}, (x_i))$ is not recorded (for some (x_i)) and replace this record by $(\mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S}, (x_i), (y_i))$; otherwise mark the record as used and send $(\texttt{Server-Sent}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S})$ to $\mathcal{S}im$. Ignore further $\texttt{Server-Send}$-message with the same $(\mathsf{ssid}, \mathcal{C}, \mathcal{S})$ from $\mathcal{S}$.

Upon receiving a query $(\texttt{Result-Send}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S})$ from the adversary $\mathcal{S}im$: ignore the message if $(\mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S}, (x_i), (y_i))$ is not recorded (for some (x_i) and (y_i)); otherwise remove the record and send $(\texttt{Result-Sent}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S}, \mathsf{IP})$ to $\mathcal{S}$ (and to $\mathcal{S}im$ if $\mathcal{S}$ is corrupted), with $\mathsf{IP} = \sum_{i=1}^{\ell} x_i \cdot y_i$. Ignore further $\texttt{Result-Send}$-message with the same $(\mathsf{ssid}, \mathcal{C}, \mathcal{S})$ from $\mathcal{S}im$.

Fig. 9: Ideal Functionality for Inner Product $\mathcal{F}_{\mathsf{IP}}$

required. This game is indistinguishable from the previous one under the $\mathsf{IND\text{-}CPA}$ property of the encryption scheme and the commitment scheme.

We remark that now, we do not need to know the exact inputs of honest players.

Game $\mathbf{G}_2$: We now deal with sessions between *a malicious client $\mathcal{C}$ and a honest server $\mathcal{S}$*. $\mathcal{S}im$ first extracts the commitment of the bits of the secret keys sk_j, and recovers sk. If these commitments do not contains bit or if these bits do not correspond to a valid secret key sk (i.e., such that the sent public key $\mathsf{pk} = g^{\mathsf{sk}}$), then $\mathcal{S}im$ chooses $K_{\mathcal{C}}$ uniformly at random. Otherwise, $\mathcal{S}im$ uses this secret key to decrypt the ciphertexts c_i for $i = 1, \ldots, \ell$, and get bits x_i. If the corresponding plaintexts are not bits, then $\mathcal{S}im$ chooses $K_{\mathcal{C}}$ uniformly at random. This game is statistically indistinguishable from the previous one, thanks to the soundness of the iZK.

Game $\mathbf{G}_3$: We now replace the CRS of the two iZK (which were generated by iSetup) by a CRS generated by TSetup and we remember the corresponding trapdoors $i\mathcal{T}$. This game is computationally indistinguishable from the previous one, thanks to the setup indistinguishability of the iZK.

Game $\mathbf{G}_4$: We now simulate all the iZK using iTKG and iTDec made by the simulator. This game is statistically indistinguishable from the previous one, thanks to the zero-knowledge property of the iZK.

Game $\mathbf{G}_5$: We now deal again with sessions between *a malicious client $\mathcal{C}$ and a honest server $\mathcal{S}$* in this game and the following ones. We replace the commitment of g^{y_i}, g^R and $g^{R'}$ by commitments of random values. This game is computationally indistinguishable from the previous one, thanks to the $\mathsf{IND\text{-}CCA}$ property of the Cramer-Shoup encryption scheme used for the commitment (and the fact that extractions are always done with different labels), and the fact that the random coins used by these commitments are no more necessary to decrypt the iZK ciphertexts $c_{\mathcal{C}}$ (thanks to the previous game).

Game $\mathbf{G}_6$: $\mathcal{S}im$ directly generates c as an encryption of $g^{R\mathsf{IP}+R'}$, by computing IP as $\sum_{i=1}^{\ell} x_i \cdot y_i$ (x_i being extracted by the encryption c_i and y_i being given as inputs to $\mathcal{S}$). This is perfectly indistinguishable to the previous game.

Game $\mathbf{G}_7$: $\mathcal{S}im$ now aborts on behalf of $\mathcal{S}$ if the last flow if not $g^{R\mathsf{IP}+R'}$ instead of just aborting when it is not such that $g^{R\mathsf{IP}'+R'}$ with $\mathsf{IP}' \in \{0, \ldots, \ell\}$. In addition, if $\mathcal{S}$ does not abort, instead of computing IP from the last flow (and so potentially getting IP'), $\mathcal{S}im$ directly outputs IP. We remark that for any IP and any fixed value for u, for any value of the last flow different than $g^{R\mathsf{IP}+R'}$, the probability this flow is $g^{R\mathsf{IP}'+R'}$ with $\mathsf{IP}' \in \{0, \ldots, \ell\}$ (so with $\mathsf{IP} \neq \mathsf{IP}'$), when R and R' are chosen uniformly at random conditioned by $u = R\mathsf{IP} + R'$, is at most ℓ/p, which is negligible. Since the adversary $\mathcal{A}$ does not know R and R' but only $R\mathsf{IP} + R'$, this game is statistically indistinguishable from the previous one.

Game $\mathbf{G}_8$: $\mathcal{S}im$ now generates c as an encryption of g^S for a random S and aborts when the last flow is not g^S. This game is perfectly indistinguishable from the previous one.

Game $\mathbf{G}_9$: $\mathcal{S}im$ now sends $(\texttt{Client-Send}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S}, (x_i))$ in behalf of $\mathcal{C}$ to the ideal functionality with (x_i) the extracted values of the malicious client $\mathcal{C}$. If $\mathcal{S}$ does not abort, $\mathcal{S}im$ also sends $(\texttt{Result-Send}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S})$ to the ideal functionality. In addition $\mathcal{S}im$ let the ideal functionality generate the output for $\mathcal{S}$. This game is perfectly indistinguishable from the previous one, since both will output the same value IP.

Game $\mathbf{G}_{10}$: We now deal again with sessions between *a honest client $\mathcal{C}$ and a malicious server $\mathcal{S}$* in this game and the following ones. $\mathcal{S}im$ now returns $g^{R\mathsf{IP}+R'}$ in the last flow (if $\mathcal{C}$ did not abort) instead of decrypting c. This game is perfectly indistinguishable from the previous one.

Game $\mathbf{G}_{11}$: In this game, we now replace the commitments of sk_j by commitments of random values. This game is computationally indistinguishable from the previous one under the $\mathsf{IND\text{-}CCA}$ property of the commitment scheme. We remark that we do not use anymore sk.

Game $\mathbf{G}_{12}$: In this game, $\mathcal{S}im$ now encrypts random values in c_i instead of the x_i's. This game is computationally indistinguishable from the previous one under the $\mathsf{IND\text{-}CPA}$ property of the commitment scheme. We remark that we do not use anymore the x_i's.

Game $\mathbf{G}_{13}$: $\mathcal{S}im$ now extracts the commitment of the bits y_i, together with g^R and $g^{R'}$. If y_i are not bits or if the ciphertext c received by $\mathcal{S}$ (under pk for which $\mathcal{S}im$ knows the secret key sk) does not contain $g^{R\mathsf{IP}+R'}$ (with $\mathsf{IP} = \sum_{i=1}^{\ell} x_i \cdot y_i$, with the extracted y_i's and the x_i given as inputs to $\mathcal{C}$), then $\mathcal{S}im$ chooses $K_{\mathcal{S}}$ uniformly at random. This game is statistically indistinguishable from the previous one, thanks to the *simulation-soundness* of the second iZK.

Game $\mathbf{G}_{14}$: $\mathcal{S}im$ now sends $(\texttt{Server-Send}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S}, (y_i))$ in behalf of $\mathcal{S}$ to the ideal functionality with (y_i) the extracted values of the malicious server $\mathcal{S}$. If $\mathcal{C}$ does not abort, $\mathcal{S}im$ also sends $(\texttt{Result-Send}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S})$ to the ideal functionality, and get the value of IP. This game is perfectly indistinguishable from the previous one, since the computed value of IP (as $\sum_{i=1}^{\ell} x_i \cdot y_i$ with (x_i) the input of $\mathcal{C}$ and (y_i) extracted from $\mathcal{S}$) is always equal to the value IP returned by the functionality.

Game $\mathbf{G}_{15}$: In last game, $\mathcal{S}im$ does not use anymore the inputs given to the honest parties. So this game is exactly the game in the ideal world.

E Other Constructions of iZK

In this appendix, we give details on the construction of iZK from Trapdoor SPHFs (TSPHFs) and from NIZK, announced in Remark 2. These constructions are here for the sake of completeness and have the disadvantage to require strong assumptions such as the random oracle model or pairing (at least for currently known over cyclic groups).

E.1 Construction of iZK from TSPHFs

With the same Language $\mathsf{i}\mathscr{L} = \mathscr{L}$ for the TSPHF and the iZK. Let us suppose we have a TSPHF (see [BBC+13] for a complete description of TSPHFs and notations we use in this section) on a language $\mathsf{i}\mathscr{L} = \mathscr{L}$. Then we can construct an iZK as follows:

- $\mathsf{iSetup}(\mathsf{crs})$ and $\mathsf{iTSetup}(\mathsf{crs})$ generates $\mathsf{icrs} := \mathsf{tcrs}$ as the common reference string for the TSPHF. The second algorithm also outputs the trapdoor $\mathsf{i}\mathscr{T}$ of the TSPHF.
- $\mathsf{iKG}(\mathsf{icrs}, \mathsf{x}, \mathsf{iw})$ outputs $(\mathsf{ipk}, \mathsf{isk}) := (\perp, \mathsf{iw})$;
- $\mathsf{iTKG}(\mathsf{i}\mathscr{T}, \mathsf{x})$ outputs $(\mathsf{ipk}, \mathsf{itk}) := (\perp, \mathsf{i}\mathscr{T})$;
- $\mathsf{iEnc}(\mathsf{ipk}, \mathsf{x})$ computes the keys $\mathsf{hk} \overset{\$}{\leftarrow} \mathsf{HashKG}(\mathsf{tcrs})$ and $\mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, \mathsf{tcrs}, \mathsf{x})$, as well as the hash value $H \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathsf{tcrs}, \mathsf{x})$. It then outputs $(c := \mathsf{hp}, K := H)$;
- $\mathsf{iDec}(\mathsf{isk} = \mathsf{iw}, c = \mathsf{hp})$ first checks hp is a valid projection key (using the algorithm VerHP for TSPHF) and aborts (by returning $\perp$) is that is not the case. It then outputs $K = \mathsf{proj}H \leftarrow \mathsf{ProjHash}(\mathsf{hp}, \mathsf{tcrs}, \mathsf{x}, \mathsf{iw})$;
- $\mathsf{iTDec}(\mathsf{itk} = \mathsf{i}\mathscr{T}, c = \mathsf{hp})$ first checks hp is a valid projection key (using the algorithm VerHP for TSPHF) and aborts (by returning $\perp$) is that is not the case. It then outputs $K = \mathsf{proj}H \leftarrow \mathsf{THash}(\mathsf{hp}, \mathsf{tcrs}, \mathsf{x}, \mathsf{i}\mathscr{T})$.

This scheme is not strictly speaking an iZK as we defined it in Section 2, since it only has computational soundness and computational zero-knowledge (in general). However, with TSPHFs based on disjunctions of two SPHFs in [ABP14], soundness and zero-knowledge would be statistical (but not with the original TSPHFs in [BBC+13]). Proofs are straightforward and are left to the reader.

For any NP Language. Using the same methods as in Appendix F, we can also get iZK for any NP language defined by a circuit (or an ABP) by considering an augmented language $\mathscr{L}$ where words also contain commitments of wires of the circuits or of intermediate values of the ABP. There exist TSPHF for these languages: the proof is similar to the one for the existence of such iZK (first an SPHF in the generic framework [BBC+13] is constructed, and then it is converted into a TSPHF).

E.2 Construction of iZK from NIZK

A TSPHF can be constructed from an SPHF and a Groth-Sahai [GS08] NIZK as explained in [BBC+13]. That gives a construction of iZK from NIZK.

F iZK for Languages Defined by a Computational Structure

F.1 Efficient iZK for Languages Represented by a Computational Structure

We have shown that a SPHF for some language $\mathscr{L}$ yields an iZK for the same language $\mathsf{i}\mathscr{L} = \mathscr{L}$. However, if the class of NP languages handled by SPHFs is sufficient for many applications, there is still a large variety of useful languages which are not captured by the framework we presented above. We thus now (informally) explain how to construct iZK for any languages just from their representation through a given computational structure.

Of course, every NP language can be represented by the most general computational structure, the *circuit*. However, more efficient, but more restricted computational structures are widely used in cryptography, such as Boolean branching programs, arithmetic formulas, etc. A computational structure of particular interest is the model of *Arithmetic Branching Programs* (ABP). They provide a very compact way to represent multivariate polynomials and capture, among others, the two structure previously given.

A language $\mathsf{i}\mathscr{L}$ represented by a computational structure can be converted into a language $\mathscr{L}$ which can be handled by the generic framework for SPHFs, by essentially extending the words with commitments to particular elements of the computational structure itself. Thus, on a given language, we can construct an iZK whose size is essentially the size of the most efficient computational structure which can represent the language.

In the following, we present the main ideas of how to construct an iZK for any NP language defined by a circuit, and also for any language defined by an ABP. We stress that they represent the most commonly used, and the most interesting, computational structures, but iZK can be constructed for others computational structures, depending of our need — other constructions exist for other representations of languages and these examples aim at illustrating the way such constructions can be made.

For any NP Language Defined by a Circuit. Let us build an iZK for an NP language $\mathscr{L}$ defined by a (polynomial-size) circuit $\mathscr{C}$ that evaluates a function F: a word x is in $\mathscr{L}$ if and only if there exists a witness iw verifying $F(\mathsf{x}, \mathsf{iw}) = 1$. We remark that any NP language can be defined by such a circuit.

The idea for the iZK construction is the following: the prover sends (as part of the public key ipk) ElGamal ciphertexts encrypting both all the bits of iw and all the values of the wires of the circuit $\mathscr{C}$ when evaluated on x and iw. Then he uses an SPHF to implicitly prove that:

- encryption of input bits of iw indeed contain bits (which is our Example 4);
- encryption of the output wire of the circuit really contains 1 (which is similar to our Example 3);
- each gate is evaluated correctly.

All these properties are guaranteed together by the conjunction of all the languages, as in our Example 5. It is thus indeed sufficient to show how to handle every individual language with the generic framework for SPHFs. The resulting scheme is an iZK for the NP language defined by $\mathscr{C}$, secure under plain DDH. It is straightforward to extend it to be secure under weaker assumptions such as DLin.

For Languages Defined by an ABP. ABP is an efficient computational model that captures, among others, the computation of Boolean formulas, Boolean branching programs and arithmetic formulas. It also gives a very compact representation of multivariate polynomials. A branching program is defined by a directed acyclic graph (V, E) with two special vertices $\mu, \nu \in V$ and a labeling function Φ. An ABP computes a function $F : \mathbb{F}_p^\ell \to \mathbb{F}_p$ (p is a prime power) as follows: Φ assigns to each edge of E either a constant value or an affine function in any number of the input variables of F, and $F(z)$ is the sum over all the path from μ to ν of the product of all the values along the path. The evaluation of F can be performed by assigning a value to each node, when nodes are sorted topologically (i.e., in such an ordering, a node appears always after its predecessors). The last node is ν and its value if the value $F(z)$.

In our case, we use ABP to define an NP language in the following way: a word x is in the language $\mathscr{L}$ if there exists a witness iw such that $F(\mathsf{x}, \mathsf{iw}) = 0$. The prover sends (as part of the public key ipk), ElGamal ciphertexts encrypting both all the bits of iw and all the values of the nodes when Φ is instantiated with x and iw. Then, as above, he uses a SPHF to implicitly prove that:

- encryption of input bits of iw indeed contain bits;
- encryption of the last node ν really contains 0;
- each value for the nodes are computed correctly; the plaintext is just the sum of the values of the previous nodes multiplied by affine evaluations on the input $(\mathsf{x}, \mathsf{iw})$.

Every individual language can be efficiently represented by an SPHF, and then conjunctions help to conclude, under the DDH assumption.

F.2 iZK for any NP Language Defined by a Circuit

In every construction described below, we consider that the additively homomorphic ElGamal encryption scheme is used. We will denote $\mathcal{E}_{\mathsf{pk}}(a; r)$ the encryption of a under the public key pk and with randomness r.

Notations. Let $F : \{0,1\}^\ell \to \{0,1\}$ be a function computed by a circuit $\mathscr{C}$ on a basis B of boolean gates (with two input wires, without loss of generality), given by its directed acyclic graph (V, E). F takes as input $z = (\mathsf{x}, \mathsf{iw})$ where $\mathsf{x} \in \{0,1\}^{\ell_\mathsf{x}}$ and $\mathsf{iw} \in \{0,1\}^{\ell_\mathsf{iw}}$ such that $\ell_\mathsf{x} + \ell_\mathsf{iw} = \ell$. Nodes or gates v in V are either an input gate corresponding to some bit x_i of x, iw_i of iw, or a constant bit, or a boolean gate in the basis B. Let $s = |V|$ be the size of the circuit. We consider the partial order on the set of gates V defined by $u \preccurlyeq v$ if there is a path from the gate u to the gate v (the graph is acyclic). Then, we index the gates $V = (v_i)_i^s$, in an order-preserving way, such that for $i = 1, \ldots, \ell$, v_i corresponds to the input bit z_i of z and, if $v_i \preccurlyeq v_j$, then $i \leq j$. For each internal gate v_i, with $i > \ell$, we denote by $\{i_1, i_2\} = \mathcal{P}(i)$ the indexes of the two preceding gates whose outputs are the inputs of v_i. The output bit of the gate v_i when evaluated on $z = (\mathsf{x}, \mathsf{iw})$ is denoted A_i (for input gates, the output bit is just the value of the input).

Extended Language for $\boldsymbol{F}$. We want an iZK for $\mathsf{i}\mathscr{L} = \{\mathsf{x} \in \{0,1\}^{\ell_\mathsf{x}} \mid \exists \mathsf{iw} \in \{0,1\}^{\ell_\mathsf{iw}}, F(\mathsf{x}, \mathsf{iw}) = 1\}$. However, this language cannot be directly handled by the SPHF framework, and we have to extend it first: we consider the extended language $\mathscr{L}$ of words of $\mathsf{i}\mathscr{L}$ along with the encryption of the output bits A_i of the gates v_i for $i > \ell_\mathsf{x}$ (hence including the input gates corresponding to the bits of iw but excluding those corresponding to the bits of x, which are anyway already known). Witness for the new language will be the random coins for all the ciphertexts, together with the values A_i for $i > \ell_\mathsf{x}$. We recall that $A_i = \mathsf{x}_i$ for $i = 1, \ldots, \ell_\mathsf{x}$.

Formally, for a gate v_i, let $(\beta_i, \beta_i^+, \beta_i^\times)$ be three integers such that, on input (x, y), the output of the gate is $(\beta_i + \beta_i^+(x + y) + \beta_i^\times xy)$. This models all the (symmetric) binary gates: $\mathsf{XOR} = (0, 1, -2)$, $\mathsf{OR} = (0, 1, -1)$, $\mathsf{AND} = (0, 0, 1)$, $\mathsf{NAND} = (1, 0, -1)$, while the unary gate NOT is just XOR 1. For $i = \ell_\mathsf{x} + 1, \ldots, s$, we consider a ciphertext $c_i = \mathcal{E}(A_i; r_i)$ of A_i with random coins r_i. We now consider the language $\mathscr{L}$ of the words $C = (\mathsf{x}, (c_i)_{i=\ell_\mathsf{iw}+1}^s)$ such that there exist witnesses $(A_i, r_i)_{i=\ell_\mathsf{iw}+1}^s$ satisfying: $A_s = 1$, for all $i = \ell_\mathsf{x} + 1, \ldots, s$, c_i encrypts the bit A_i with random coins r_i, and, for $i = \ell, \ldots, s$, A_i

Algorithm 1 Dynamic ABP Computation

```
1: procedure DAC($F, x$)                                  ▷ $F$ is an ABP and $x$ is its input
2:     $A_0 \leftarrow 1$
3:     for $i = 1$ to $|V|$ do
4:         $A_i \leftarrow 0$
5:         for all $v_j \in \mathrm{prec}(v_i)$ do
6:             $A_i \leftarrow A_i + \Phi((v_j \to v_i), x) \cdot A_j$      ▷ $A_0$ is set as the value of the predecessor of $v_1$
7:     return $(A_i)_{2 \le i \le |V|}$                                  ▷ $A_{|V|} = F(x)$
```

verifies the appropriate relation with A_{i_1} and A_{i_2}, for $\{i_1, i_2\} = \mathcal{P}(i)$. However, there are quadratic relations, we thus need additional variables to linearize the system.

Now, let us show how to construct an SPHF on this language $\mathscr{L}$ which can be automatically used to construct an iZK using the framework defined in Section 3 for the above language i$\mathscr{L}$. Concretely, we use an ElGamal encryption in basis g, with public key h, and we write $c_i = (c_i^1 = g^{r_i}, c_i^2 = h^{r_i} g^{A_i})$. $C = (\mathsf{x}, (c_i)_{i=\ell_{\mathsf{iw}}+1}^s)$ is in $\mathscr{L}$ if and only if there exist $(A_i)_{i=\ell_{\mathsf{x}}+1}^s \in \{0,1\}^{s-\ell_{\mathsf{x}}}$, $(r_i)_{i=\ell_{\mathsf{x}}+1}^s \in \mathbb{Z}_p^{s-\ell_{\mathsf{x}}}$, $(\mu_i)_{i=\ell_{\mathsf{x}}+1}^s \in \mathbb{Z}_p^{s-\ell_{\mathsf{x}}}$ and $(\mu'_i)_{i=\ell+1}^s \in \mathbb{Z}_p^{s-\ell}$, such that:

$$\begin{array}{ccc} g^{r_i} = c_i^1 & \text{and} & h^{r_i} \cdot g^{A_i} = c_i^2 \\ (c_i^1)^{A_i} \cdot g^{-\mu_i} = 1 & \text{and} & (c_i^2/g)^{A_i} \cdot h^{-\mu_i} = 1 \end{array}$$

for $i = \ell_{\mathsf{x}} + 1, \ldots, s$ and:

$$(c_{i_2}^1)^{A_{i_1}} \cdot g^{-\mu'_i} = 1 \quad \text{and} \quad g^{\beta_i^+ A_{i_1}} \cdot g^{\beta_i^+ A_{i_2}} \cdot (c_{i_2}^2)^{\beta_i^\times A_{i_1}} \cdot h^{-\beta_i^\times \mu'_i} \cdot g^{-A_i} = g^{-\beta_i} \quad \text{for } i = \ell+1, \ldots, s$$

for $i = \ell+1, \ldots, s$, with $\{i_1, i_2\} = \mathcal{P}(i)$, since the second of equations ensures $\mu_i = r_i A_i$ and $A_i(A_i - 1) = 0$ (i.e., A_i is a bit), while the third one ensures $\mu'_i = r_{i_2} A_{i_1}$ and $A_i = \beta_i + \beta_i^+(A_{i_1} + A_{i_2}) + \beta_i^\times A_{i_1} A_{i_2}$. These linear equations (in the exponents) directly provides the matrix $\Gamma(C)$, while $\theta(C)$ is defined by the right-hand sides of the relations. This then leads to an SPHF over $\mathscr{L}$, based on the plain DDH.

F.3 iZK for any NP Language Defined by an ABP

Notations: Let $F : \mathbb{Z}_p^\ell \to \mathbb{Z}_p$ be a function computed by an ABP given by its directed acyclic graph (V, E), two special vertices $\mu, \nu \in V$ and a labeling function $\Phi : E \times \mathbb{Z}_p^\ell \to \mathbb{Z}_p$. F takes as input $z = (\mathsf{x}, \mathsf{iw})$, where $\mathsf{x} \in \mathbb{Z}_p^{\ell_{\mathsf{x}}}$ and $\mathsf{iw} \in \mathbb{Z}_p^{\ell_{\mathsf{iw}}}$ such that $\ell_{\mathsf{x}} + \ell_{\mathsf{iw}} = \ell$. Let $s = |E|$ be the size of the ABP. We denote by $(u \to v)$ the edge from the vertex u to the vertex v. We consider the partial order on the set of vertices V defined by $u \preccurlyeq v$ if there is a path from the gate u to the gate v (the graph is acyclic). Then, we index the vertices $V = (v_i)_s$ in an order-preserving way: $v_i \preccurlyeq v_j \Rightarrow i \le j$, $\mu = v_1$ and $\nu = v_{|V|}$. For each node $v \neq \mu$, we denote by $\mathrm{prec}(v)$ the set of direct predecessors of v, i.e., the vertices u such that $(u \to v) \in E$. Algorithm 1 describes the way the ABP is evaluated in an input x. When the input x can be seen as a pair of tuples $x = (\mathsf{x}, \mathsf{iw}) \in \mathbb{Z}_p^{\ell_{\mathsf{x}}} \times \mathbb{Z}_p^{\ell_{\mathsf{iw}}} = \mathbb{Z}_p^\ell$, we consider the problem, for a given x, of the existence of a witness iw such that $F(\mathsf{x}, \mathsf{iw}) = 0$. We want to build an iZK on the language of the words x with such witnesses iw.

Extended Language for *F*. As above, we want an iZK for i$\mathscr{L} = \{x \in \mathbb{Z}_p^{\ell_{\mathsf{x}}} \mid \exists \mathsf{iw} \in \mathbb{Z}_p^{\ell_{\mathsf{iw}}}, F(\mathsf{x}, \mathsf{iw}) = 0\}$. We can extend it, as above, with the ciphertexts c_i of all the witnesses iw_i and a_i of all intermediate values A_i of the dynamic ABP computation (except the special vertices $A_1 = 1$ and $A_{|V|} = 0$). Then the witnesses are $(r_i)_{i=1}^{\ell_{\mathsf{iw}}}$ and $(s_i)_{i=2}^{|V|-1}$, the random coins for the encryption. We now consider the language $\mathscr{L}$ of the words $(\mathsf{x}, (c_i)_{i=1}^{\ell_{\mathsf{iw}}}, (a_i)_{i=2}^{|V|-1})$ such that there exist witnesses $((r_i, \mathsf{iw}_i)_{i=1}^{\ell_{\mathsf{iw}}}, (s_i, A_i)_{i=2}^{|V|-1})$ satisfying: for all $i = 1, \ldots \ell_{\mathsf{iw}}$, c_i encrypts the scalar iw_i with random coins r_i, for $i = 2, \ldots, |V| - 1$, a_i encrypts the scalar A_i with random coins s_i, and A_i verifies the appropriate relation w.r.t. its predecessors, as well $A_1 = 1$ and $A_|V| = 0$, which introduces again quadratic relations.

As above, using ElGamal encryption, we can write $c_i = (c_i^1 = g^{r_i}, c_i^2 = h^{r_i} g^{\mathsf{iw}_i})$ and $a_i = (a_i^1 = g^{s_i}, a_i^2 = h^{s_i} g^{A_i})$. The word $C = (\mathsf{x}, (c_i)_{i=1}^{\ell_{\mathsf{iw}}}, (a_i)_{i=2}^{|V|-1})$ is in $\mathscr{L}$ if and only if there exist $(\mathsf{iw}_i)_{i=1,\ldots,\ell_{\mathsf{iw}}} \in \mathbb{Z}_p^{\ell_{\mathsf{iw}}}$,

$(r_i)_{i=1,\dots,\ell_{\mathsf{iw}}} \in \mathbb{Z}_p^{\ell_{\mathsf{iw}}}$, $(A_i)_{i=2}^{|V|-1} \in \mathbb{Z}_p^{|V|-2}$, $(s_i)_{i=2}^{|V|-1} \in \mathbb{Z}_p^{|V|-2}$, and $\mu_{i,j} \in \mathbb{Z}_p$, for $i = 2, \dots, |V|-1$ and $j = 1, \dots, \ell_{\mathsf{iw}}$ (but actually for all the values iw_j that appears in labels on edges leaving from v_i), such that:

$$\begin{aligned}
g^{r_i} &= c_i^1 \text{ and } h^{r_i} \cdot g^{\mathsf{iw}_i} = c_i^2 && \text{for } i = 1, \dots, \ell_{\mathsf{x}} \\
g^{s_i} &= a_i^1 \text{ and } h^{s_i} \cdot g^{A_i} = a_i^2 && \text{for } i = 2, \dots, |V|-1 \\
(a_i^1)^{\mathsf{iw}_j} \cdot g^{-\mu_{i,j}} &= 1 && \text{for } i = 2, \dots, |V|-1, \text{ for } j = 1, \dots, \ell_{\mathsf{iw}} \\
g^{\sum_{v_j \in \mathtt{prec}(v_i)} A_i \cdot \Phi((v_j \to v_i), x) - A_i} &= 1 && \text{for } i = 2, \dots, |V|, \text{ with } A_{|V|} = 0
\end{aligned}$$

where $x = (\mathsf{x}, \mathsf{iw})$. We recall that $\Phi(v_j \to v_i)$ is an affine function (or a constant) in x (known by both players) and iw (encrypted in the c_i's). So quadratic terms $A_i \mathsf{iw}_j$ can be computed using the intermediate value $\mu_{i,j}$, as above, that implicitly corresponds to $r_i \mathsf{iw}_j$ to remove extra terms in h introduced by $(a_i^2)^{\mathsf{iw}_j}$: $(a_i^2)^{\mathsf{iw}_j} \cdot h^{-\mu_{i,j}}$ is indeed $g^{A_i \mathsf{iw}_j}$ when the first row is enforced.

A Concrete Example. Let us consider the following language $\mathsf{i}\mathscr{L} = \{(\mathsf{x}_1, \mathsf{x}_2) \in \mathbb{Z}_p^2 \mid \exists \mathsf{iw} \in \mathbb{Z}_p, (\mathsf{iw}^2 - \mathsf{x}_1)(\mathsf{iw}^2 - \mathsf{x}_2) = 0\}$ of pairs of integers modulo p such that at least one of the elements of the pair is a square. This language can be efficiently represented by the following ABP:

$$\begin{tikzcd} {} \arrow[r] & v_1 \arrow[r, bend left, no head, "\mathsf{iw}"] \arrow[rr, bend right, no head, "-\mathsf{x}_1"'] & v_2 \arrow[r, bend left, no head, "\mathsf{iw}"] & v_3 \arrow[r, bend left, no head, "\mathsf{iw}"] \arrow[rr, bend right, no head, "-\mathsf{x}_2"'] & v_4 \arrow[r, bend left, no head, "\mathsf{iw}"] & v_5 \arrow[r] & {} \end{tikzcd}$$

Applying the dynamic ABP computation algorithm, we get $A_1 = 1$, $A_2 = \mathsf{iw}$, $A_3 = \mathsf{iw}A_2 - \mathsf{x}_1 A_1 = \mathsf{iw}^2 - \mathsf{x}_1$, $A_4 = \mathsf{iw}A_3$, and $A_5 = (\mathsf{iw}^2 - \mathsf{x}_2)A_3 = (\mathsf{iw}^2 - \mathsf{x}_1)(\mathsf{iw}^2 - \mathsf{x}_2)$. Thus, we construct the extended language of words $\mathscr{L}' = \{(\mathsf{x}_1, \mathsf{x}_2), (c^1, c^2), (a_i^1, a_i^2)_{i=2}^4\}$ such that there exists $(\mu_i)_{i=2}^4 \in \mathbb{Z}_p^3$ so that the plaintexts iw, and (A_2, A_3, A_4) satisfy:

$$\begin{aligned}
g^r &= c^1 \text{ and } h^r \cdot g^{\mathsf{iw}} = c^2 \\
g^{s_i} &= a_i^1,\ h^{s_i} \cdot g^{A_i} = a_i^2 \text{ and } (a_i^1)^{\mathsf{iw}} \cdot g^{-\mu_i} = 1 && \text{for } i = 2, 3, 4 \\
g^{\mathsf{iw}} \cdot g^{-A_2} &= 1 \text{ and } (a_2^2)^{\mathsf{iw}} \cdot h^{-\mu_2} \cdot g^{-A_3} = g^{\mathsf{x}_1} \\
(a_3^2)^{\mathsf{iw}} \cdot h^{-\mu_3} \cdot g^{-A_4} &= 1 \text{ and } (a_4^2)^{\mathsf{iw}} \cdot h^{-\mu_4} = g^{\mathsf{x}_2}
\end{aligned}$$

We have 15 equations and 11 witnesses $(\mathsf{iw}, r, (A_i)_2^4, (s_i)_2^4, (\mu_i = s_i \mathsf{iw})_2^4)$. However, in this particular example, we can drop three equations and two witnesses: as the value A_2 is exactly the witness iw, we can use drop (c^1, c^2) and use (a_2^1, a_2^2) instead. In addition, we can remove the two first equations and the equation $g^{\mathsf{iw}} \cdot g^{-A_2} = 1$: we now have 12 equations and 10 witnesses $((A_i)_2^4, (s_i)_2^4, (\mu_i = s_i \mathsf{iw})_2^4)$. We stress the fact that in particular applications with a "good" structure, it is often possible to get optimizations on the theoretical size of the corresponding iZK. This leads to a iZK with public key of size $|\mathsf{ipk}| = 30$ and ciphertexts of size $|c| = 24$ (using the optimization of C).