Non-Interactive Zero-Knowledge Proofs of Non-Membership

Olivier Blazy[1] and Céline Chevalier[2] and Damien Vergnaud[3]

[1] Université de Limoges, XLim, France
[2] Université Paris II, France
[3] ENS, Paris, France [†]

Abstract. Often, in privacy-sensitive cryptographic protocols, a party commits to a secret message m and later needs to prove that m belongs to a language $\mathcal{L}$ or that m does *not* belong to $\mathcal{L}$ (but does not want to reveal any further information). We present a method to prove in a non-interactive way that a committed value does not belong to a given language $\mathcal{L}$. Our construction is generic and relies on the corresponding proof of membership to $\mathcal{L}$. We present an efficient realization of our proof system by combining smooth projective hash functions and Groth-Sahai proof system.

In 2009, Kiayias and Zhou introduced zero-knowledge proofs with witness elimination which enable to prove that a committed message m belongs to a set $\mathcal{L}$ in such a way that the verifier accepts the interaction only if m does not belong to a set determined by a public relation Q and some *private* input m' of the verifier. We show that the protocol they proposed is flawed and that a dishonest prover can actually make a verifier accept a proof for any message $m \in L$ even if $(m, m') \in Q$. Using our non-interactive proof of non-membership of committed values, we are able to fix their protocol and improve its efficiency.

Our approach finds also efficient applications in other settings, e.g. in anonymous credential systems and privacy-preserving authenticated identification and key exchange protocols.

Keywords. Zero Knowledge, Witness Elimination, Smooth Projective Hash Function, Groth-Sahai Proof System

1 Introduction

In cryptography, when designing privacy-sensitive applications, the use of commitments and corresponding zero-knowledge proofs is often indispensable. They allow a prover to convince a verifier that a digitally committed value is a member of a given language (without revealing any further information beyond this membership). An important instance of this problem consists in showing that the committed value lies in a given finite set (*e.g.* in e-auctions or e-voting protocols, a bidder or voter has to prove that his secret bid or vote is chosen from a list of candidates, see [CCs08] and references therein). However one usually wants to demonstrate more complex properties about committed values. For instance in anonymous credentials systems and privacy-preserving authenticated identification or key exchange protocols, a participant must usually prove the possession of a credential issued by an authority (without revealing it).

For the latter primitives, it is often necessary to prove combination of simple statements about several credentials issued by the authority (OR, AND, and NOT connectives) [CG08,ILV11]. For instance, a crucial requirement is that credentials issued can be later revoked. In principle, revocation lists can be used for anonymous credentials by having the user to prove in zero-knowledge that his credential is *not* contained in the list. However, this is usually inefficient since the computational and communication costs grow with the number of entries in the list. Recently, Bayer and Groth [BG13] proposed an efficient interactive solution for blacklisting anonymous users (with logarithmic growth) but their elegant technique does not generalize readily to prove the non-membership to arbitrary languages.

In these scenarios, it is usually desired that the zero-knowledge proofs are non-interactive. For example, in the e-voting scenario, the membership proof is a part of the vote validity proof that is verified by various parties without any active participation of the voter. In this paper, we present a generic method to prove in a non-interactive way that a committed value does not belong to a given language. Our approach finds efficient applications in various settings, e.g. in zero-knowledge with witness elimination [KZ09] or language authenticated key exchange [BBC$^+$13a].

[†] CNRS – UMR 8548 and INRIA – EPI Cascade

1.1 Related work

A commitment scheme allows a user to commit to a message m by publishing a commitment C, and this commitment can be opened at a later point in time. It can be seen as the digital analogue of a "sealed envelope": the security properties required are called the *hiding* property (one cannot learn anything about the message m from the commitment C) and *binding* property (one cannot open the commitment C to a different message $m' \neq m$). Zero-knowledge proofs of knowledge are two party protocols, which allow a prover to convince a verifier that he knows some secret piece of information, without the verifier being able to learn anything about the secret value (except for what is revealed by the claim itself). Often in cryptographic protocols, a party chooses a message m and then commits to it. He keeps the message secret and publishes the commitment. He later needs to prove that m belongs to a finite set $\mathcal{L}$ or that m does *not* belong to $\mathcal{L}$, but cannot reveal anything about m.

For a finite set $\mathcal{L}$ with no additional structure, the most efficient combination of commitment and zero-knowledge proof was recently proposed by Bayer and Groth [BG13]. The interactive proof system is quite efficient: it has $O(\log(\#\mathcal{L}))$ communication and computational complexity and significantly improves the previous proposals with $O(\sqrt{\#\mathcal{L}})$ complexity [Pen11]. It can be made non-interactive in the random oracle model by using the Fiat-Shamir heuristic[1].

There also exist efficient membership proofs for families of very large sets $\mathcal{L}$ equipped with an "algebraic structure" (e.g. the set of valid message/digital signatures pairs for a given public key whose cardinal is exponential in the security parameter). Most of them also admit efficient non-membership proof systems. However, up to now there is no generic construction and these zero-knowledge proofs of non-membership of committed values require specific security analysis.

1.2 Contributions of the paper

The first contribution of the paper is to present an efficient non-interactive technique to prove (in zero-knowledge) that a committed message does not belong to a set $\mathcal{L}$. The proof is generic and relies on a proof of membership to $\mathcal{L}$ with specific mild properties. In particular, it is independent of the size of $\mathcal{L}$ and if there exists an efficient proof of membership for committed values, one gets readily an efficient proof of non-membership. Instantiated with a combination of *smooth projective hash functions* and Groth-Sahai proof system, we obtain very efficient realization for non-interactive proof of non-membership of committed values.

In 2009, Kiayias and Zhou [KZ09] introduced *zero-knowledge proofs with witness elimination*. This primitive enables to prove that a committed message m belongs to a set $\mathcal{L}$ (with a witness w) in such a way that the verifier accepts the interaction only if w does not belong to a set determined by a public relation Q and some *private* input w' of the verifier. The verifier does not learn anything about w (except that $m \in \mathcal{L}$ and $(w, w') \notin Q$) and the prover does not learn anything about w'. The primitive can obviously be used to handle revocation lists. It was motivated in [KZ09] by privacy-preserving identification schemes when a user wishes to authenticate himself to a verifier while preserving his anonymity and the verifier makes sure the prover does not match the identity of a suspect user that is tracked by the authorities (without leaking any information about the suspect identity).

We show that the original proposal of zero-knowledge proofs with witness elimination from [KZ09] is flawed and that a dishonest prover can actually make a verifier accept a proof for any message $m \in L$ even if $(w, w') \in Q$. In particular, in the suspect tracking scenario, a dishonest prover can identify himself even if he is on the suspect list. Therefore, their protocol does not achieve the claimed security. However we explain how to apply our proof of non-membership to fix it. We obtain a proof system that achieves the security goal and is more efficient than the original (insecure) solution.

Eventually, we briefly present applications of our proof of non-membership to other settings such as anonymous credentials and privacy-preserving authenticated key exchange.

[1] It is worth noting that there exist protocols that are secure in the random oracle models and insecure in the plain model [CGH04].

2 Preliminaries

In this section we recall various classical definitions, tools used throughout this paper. We use classical definitions and notations and the familiar reader may skip this section.

2.1 Definitions

Encryption An encryption scheme $\mathcal{E}$ is described through four algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$:

- $\mathsf{Setup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}(\mathsf{param})$ outputs a pair of keys, a (public) encryption key ek and a (private) decryption key dk;
- $\mathsf{Encrypt}(\mathsf{ek}, M; \rho)$ outputs a ciphertext $\mathcal{C}$, on the message M, under the encryption key ek, with randomness ρ;
- $\mathsf{Decrypt}(\mathsf{dk}, \mathcal{C})$ outputs the plaintext M, encrypted in the ciphertext $\mathcal{C}$ or $\perp$.

Such encryption scheme is required to have the classical properties, *Correctness* and *Indistinguishability under Chosen Plaintext Attack [GM84]*:

- *Correctness*: For every pair of keys $(\mathsf{ek}, \mathsf{dk})$ generated by KeyGen, every messages M, and every random ρ, we should have $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{Encrypt}(\mathsf{ek}, M; \rho)) = M$.
- *Indistinguishability under Chosen Plaintext Attack [GM84]*: This notion (IND-CPA), formalized by the adjacent game, states that an adversary should not be able to efficiently guess which message has been encrypted even if he chooses the two original plaintexts.
The advantages are:

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

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(\mathfrak{K}, t) = \max_{\mathcal{A} \leq t} \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}).$$

```
$\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$
1.$\mathsf{param} \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2.$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
3.$(M_0, M_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{ek})$
4.$c^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}, M_b)$
5.$b' \leftarrow \mathcal{A}(\texttt{GUESS} : c^*)$
6.RETURN $b'$
```

Zero-Knowledge Proofs Classical definitions and notations for *non-interactive* zero-knowledge proof systems are given in appendix A.

2.2 Classical Hypotheses

A bilinear group is a tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of prime order p, generated respectively by g_1,g_2 and $e(g_1, g_2)$ and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerated bilinear form, *i.e.*:

$$\forall X \in \mathbb{G}_1, \forall Y \in \mathbb{G}_2, \forall \lambda, \mu \in \mathbb{Z}_p : e(X^\lambda, Y^\mu) = e(X, Y)^{\lambda\mu}$$

and $e(g_1, g_2)$ does indeed generate the prime order group $\mathbb{G}_T$. In the following we will suppose there exists a polynomial time algorithm which outputs such bilinear groups.

In this paper, we will present concrete instantiation based on standard problems on groups:
*Decisional Diffie Hellman (*DDH*)* [Bon98]: The Decisional Diffie-Hellman hypothesis states that in a group $(p, \mathbb{G}, g)$ (written in multiplicative notation), given (g^μ, g^ν, g^ψ) for unknown $\mu, \nu \overset{\$}{\leftarrow} \mathbb{Z}_p$, it is hard to decide whether $\psi = \mu\nu$.
*Symmetric External Diffie Hellman (*SXDH*)* [ACHdM05]: this variant used in bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, states that DDH is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.3 Classical Tools

Smooth Projective Hash Functions [CS02] Smooth projective hash functions (SPHF) were introduced by Cramer and Shoup [CS02]. A projective hashing family is a family of hash functions that can be evaluated in two ways: using the (secret) hashing key, one can compute the function on every point in its domain, whereas using the (public) *projected* key one can only compute the function on a special subset of its domain. Such a family is deemed *smooth* if the value of the hash function on any point outside the special subset is independent of the projected key. The notion of SPHF has found applications in various contexts in cryptography (*e.g.* [GL03,Kal05,ACP09,BPV12]).

Smooth Projective Hashing System: A Smooth Projective Hash Function over a language $\mathfrak{L} \subset X$, onto a set $\mathbb{G}$, is defined by five algorithms $(\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$:

- $\mathsf{Setup}(1^{\mathfrak{K}})$ where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme, and the description of an $\mathcal{NP}$ language $\mathfrak{L}$;
- $\mathsf{HashKG}(\mathfrak{L}, \mathsf{param})$, outputs a hashing key hk for the language $\mathfrak{L}$;
- $\mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W)$, derives the projection key hp, possibly depending on the word W [GL03,ACP09] thanks to the hashing key hk.
- $\mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W)$, outputs a hash value $v \in \mathbb{G}$, thanks to the hashing key hk, and W
- $\mathsf{ProjHash}(\mathsf{hp}, (\mathfrak{L}, \mathsf{param}), W, w)$, outputs the hash value $v' \in \mathbb{G}$, thanks to the projection key hp and the witness w that $W \in \mathfrak{L}$.

In the following, we consider $\mathfrak{L}$ as a hard-partitioned subset of X, *i.e.* it is computationally hard to distinguish a random element in $\mathfrak{L}$ from a random element in $X \setminus \mathfrak{L}$.

A Smooth Projective Hash Function SPHF should satisfy the following properties:

- *Correctness*: Let $W \in \mathfrak{L}$ and w a witness of this membership. Then, for all hashing keys hk and associated projection keys hp we have $\mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W) = \mathsf{ProjHash}(\mathsf{hp}, (\mathfrak{L}, \mathsf{param}), W, w)$.
- *Smoothness*: For all $W \in X \setminus \mathfrak{L}$ the following distributions are statistically indistinguishable:

$$\left\{ (\mathfrak{L}, \mathsf{param}, W, \mathsf{hp}, v) \;\middle|\; \begin{array}{l} \mathsf{param} = \mathsf{Setup}(1^{\mathfrak{K}}), \\ \mathsf{hk} = \mathsf{HashKG}(\mathfrak{L}, \mathsf{param}), \\ \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W), \\ v = \mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W) \end{array} \right\} \simeq \left\{ (\mathfrak{L}, \mathsf{param}, W, \mathsf{hp}, v) \;\middle|\; \begin{array}{l} \mathsf{param} = \mathsf{Setup}(1^{\mathfrak{K}}), \\ \mathsf{hk} = \mathsf{HashKG}(\mathfrak{L}, \mathsf{param}), \\ \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W), \\ v \overset{\$}{\leftarrow} \mathbb{G} \end{array} \right\}.$$

- *Pseudo-Randomness*: If $W \in \mathfrak{L}$, then without a witness of membership the two previous distributions should remain computationally indistinguishable.

The article [BBC$^+$13b] introduced a new notation for SPHF: For a language $\mathfrak{L}$, we assume there exist a function Γ and a family of functions Θ, such that $\boldsymbol{u} \in \mathfrak{L}$, if and only if, $\Theta(\boldsymbol{u})$ is a linear combination of the rows of $\Gamma(\boldsymbol{u})$. We furthermore require that a user, who knows a witness w of the membership $\boldsymbol{u} \in \mathfrak{L}$, can efficiently compute the linear combination $\boldsymbol{\lambda}$.

With the above notations, the hashing key is a vector $\mathsf{hk} = \boldsymbol{\alpha}$, while the projection key is, for a word $\boldsymbol{u}$, $\mathsf{hp} = \boldsymbol{\gamma}(\boldsymbol{u}) = \Gamma(\boldsymbol{u}) \odot \boldsymbol{\alpha}$ (where $\odot$ denotes the Hadamard product, i.e. the entry-wise product). Then, the hash value is: $\mathsf{Hash}(\mathsf{hk}, \boldsymbol{u}) \overset{\text{def}}{=} \Theta(\boldsymbol{u}) \odot \boldsymbol{\alpha} = \boldsymbol{\lambda} \odot \boldsymbol{\gamma}(\boldsymbol{u}) \overset{\text{def}}{=} \mathsf{ProjHash}(\mathsf{hp}, \boldsymbol{u}, w)$.

Groth-Sahai Proof System. Groth and Sahai [GS08] proposed non-interactive zero-knowledge proofs of satisfiability of certain equations over bilinear groups, called *pairing product equations*. Using as witness group elements (and scalars) which satisfy the equation, the prover starts with making commitments on them. The commitment key is of the form $\mathbf{u}_1 = (u_{1,1}, u_{1,2}), \mathbf{u}_2 = (u_{2,1}, u_{2,2}) \in \mathbb{G}_1^2$ and $\mathbf{v}_1 = (v_{1,1}, v_{1,2}), \mathbf{v}_2 = (v_{2,1}, v_{2,2}) \in \mathbb{G}_2^2$. We write

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \begin{pmatrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \end{pmatrix} \quad \text{and} \quad \mathbf{v} = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{pmatrix} = \begin{pmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \end{pmatrix}.$$

The Setup algorithm initializes the parameters as follows: $\mathbf{u}_1 = (g_1, u)$ with $u = g_1^\lambda$ and $\mathbf{u}_2 = \mathbf{u}_1{}^\mu$ with $\lambda, \mu \xleftarrow{\$} \mathbb{Z}_p^*$, which means that $\mathbf{u}$ is a Diffie-Hellman tuple in $\mathbb{G}_1$, since $\mathbf{u}_1 = (g_1, g_1^\lambda)$ and $\mathbf{u}_2 = (g_1^\mu, g_1^{\lambda\mu})$. The TSetup algorithm will use instead $\mathbf{u}_2 = \mathbf{u}_1{}^\mu \odot (1, g_1)^{-1}$: $\mathbf{u}_1 = (g_1, g_1^\lambda)$ and $\mathbf{u}_2 = (g_1^\mu, g_1^{\lambda\mu-1})$. And it is the same in $\mathbb{G}_2$ for $\boldsymbol{v}$. Depending on the definition of $\mathbf{u}_2$, $\mathbf{v}_2$, this commitment can be either perfectly hiding or perfectly binding. The two parameter initializations are indistinguishable under the SXDH assumption.

To commit to $X \in \mathbb{G}_1$, one chooses randomness $s_1, s_2 \in \mathbb{Z}_p$ and sets $\mathcal{C}(X) = (1, X) \odot \mathbf{u}_1^{s_1} \odot \mathbf{u}_2^{s_2} = (u_{1,1}^{s_1} \cdot u_{2,1}^{s_2}, X \cdot u_{1,2}^{s_1} \cdot u_{2,2}^{s_2})$. Similarly, one can commit to element in $\mathbb{G}_2$ and scalars in $\mathbb{Z}_p$. The committed group elements can be extracted if $\mathbf{u}_2$ is linearly dependant of $\mathbf{u}_1$ by knowing the discrete logarithm x_1 between $\mathbf{u}_{1,1}$ and $\mathbf{u}_{2,2}$: $c_2/(c_1^{x_1}) = X$.

To prove satisfiability of an equation (which is the statement of the proof), a Groth-Sahai proof uses these commitments and shows that the committed values satisfy the equation. The proof consists again of group elements and is verified by a pairing equation derived from the statement.

We refer to [GS08] for details of the Groth-Sahai proof system.

3 Proof of No-Statement

3.1 Generic Technique

In this section, we are going to present a way to prove exclusion statement, following a Commit and Prove approach ([CLOS02]).

The underlying idea is that, we are going to try to build a proof of validity for the statement (which is supposed to not be verified), and prove that we are failing to do so while being completely honest. Hence once we prove that the proof is correctly generated, the fact that the verification fails means that the initial statement did not hold (under the completeness of the proof).

Let us consider a language $\mathcal{L}$. We assume that it is easy to test whether a word w belongs to $\mathcal{L}$ (in probabilistic polynomial time).

Let $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be an encryption scheme and let $(\mathsf{ek}, \mathsf{dk})$ be a pair of keys output by $\mathsf{KeyGen}(\mathsf{param})$ (where param are global parameters output by $\mathsf{Setup}(1^\mathfrak{K})$). We assume that the encryption scheme is *without redundancy* [PP03]: *i.e.* all ciphertexts are valid (which means here "reachable"), the encryption function is not only a probabilistic injection, but also a surjection. A prover possesses a word w not in a language $\mathcal{L}$ which is encrypted in $\mathcal{C}$ using some randomness r, $\mathcal{C} = \mathsf{Encrypt}(\mathsf{ek}, w; r)$. The prover wants to prove that C encrypts a word that does *not* belong to $\mathcal{L}$ in zero-knowledge. To construct our proof system, we are going to follow a generic approach by combining two different proof systems (in our following instantiations the first proof will be a SPHF, while the second one will be a Groth Sahai NIZK).

Formally, we assume there is a sound and correct non-interactive (NI) proof system $\mathit{\Pi}_a$ for the language defined by the binary relation $\mathcal{R} = \{(\mathcal{C}, (w, r)), C = \mathsf{Encrypt}(\mathsf{ek}, w; r) \wedge w \in \mathcal{L}\}$ and we will construct a NI zero-knowledge proof system for the language defined by the binary relation $\hat{\mathcal{R}} = \{(\mathcal{C}, (w, r)), C = \mathsf{Encrypt}(\mathsf{ek}, w; r) \wedge w \notin \mathcal{L}\}$.

We assume that $\mathit{\Pi}_a$ satisfies the following properties:

- there exists a randomization algorithm that takes a proof π_a output by $\mathit{\Pi}_a.\mathsf{Prove}(\mathcal{C}, (w, r))$ and some randomness r', ρ' and outputs a properly distributed proof π'_a on the same word w encrypted in the ciphertext $\mathcal{C}'$ using randomness r'[2].

[2] The idea is that there is no "weak" randomness, and so if the adversary breaks the completeness of the proof for a pair w, ρ, any pair w, ρ' leads to invalid proofs even in w is in $\mathcal{L}$. This property is easy to achieve with homomorphic proof system, Groth Sahai and SPHF are good candidates for example.

- Π_a verifies the *indistinguishability of proofs* property: given a pair $(\mathcal{C}, (w, r)) \in \hat{R}$ where $\mathcal{C}$ is an encryption of a word $w \notin \mathcal{L}$ using randomness r, it should be hard to distinguish an invalid proof generating *honestly* as $\Pi_a.\mathsf{Prove}(\mathcal{C}, (w, r))$ from a random value.[3]
- there exists a NI zero-knowledge proof system Π_b where given an output π_a proves that it is indeed the correct result from $\Pi_a.\mathsf{Prove}(\mathcal{C}, (w, r); \rho)$ even if $w \notin \mathcal{L}$. We also require the extra property, that either Π_b has perfect soundness or possesses a trapdoor allowing to recover ρ.

Assuming these three properties, our NI zero-knowledge proof system for the language defined by the binary relation $\hat{\mathcal{R}}$, $\Pi = (\mathsf{no.Setup}, \mathsf{no.TSetup}, \mathsf{no.Prove}, \mathsf{no.Verify}, \mathsf{no.Simulate})$, is defined as follows:

$\mathsf{no.Setup}(1^{\mathfrak{K}})$**:** runs $\mathcal{E}.\mathsf{Setup}(1^{\mathfrak{K}})$ to compute the global parameters param and $\mathsf{KeyGen}(\mathsf{param})$ to obtain a key pair $(\mathsf{ek}, \mathsf{dk})$. It also runs $\Pi_a.\mathsf{Setup}(1^{\mathfrak{K}})$ to obtain crs_a and $\Pi_b.\mathsf{Setup}(1^{\mathfrak{K}})$ to obtain crs_b. It outputs the common reference string $\mathsf{crs} = (\mathsf{crs}_a, \mathsf{crs}_b, \mathsf{param}, \mathsf{ek})$.

$\mathsf{no.TSetup}(1^{\mathfrak{K}})$**:** runs $\mathcal{E}.\mathsf{Setup}(1^{\mathfrak{K}})$ to compute the global parameters param and $\mathsf{KeyGen}(\mathsf{param})$ to obtain a key pair $(\mathsf{ek}, \mathsf{dk})$. It also runs $\Pi_a.\mathsf{Setup}(1^{\mathfrak{K}})$ to obtain crs_a and $\Pi_b.\mathsf{TSetup}(1^{\mathfrak{K}})$ to obtain (crs_b, τ_b). It outputs the common reference string $\mathsf{crs} = (\mathsf{crs}_a, \mathsf{crs}_b, \mathsf{param}, \mathsf{ek})$ and the trapdoor $\tau = (\mathsf{dk}, \tau_b)$.

$\mathsf{no.Prove}(\mathcal{C}, w, s, \mathsf{crs}; \rho)$**:** Computes $\pi_a := \Pi_a.\mathsf{Prove}(\mathcal{C}, (w, r); \rho)$, and a proof $\pi_b := \Pi_b.\mathsf{Prove}(\pi_a, \mathcal{C}, (w, r), \rho; \rho')$, and outputs (π_a, π_b)

$\mathsf{no.Verify}(\mathcal{C}, \pi_a, \pi_b)$**:** returns 1 if and only if π_a is invalid for Π_a and π_b is valid for Π_b ((*i.e.* $\Pi_a.\mathsf{Verify}(\mathcal{C}, \pi_a) = 0$ and $\Pi_b.\mathsf{Verify}(\pi_b, \pi_a, \mathcal{C}) = 1$).

$\mathsf{no.Simulate}(\mathcal{C}, \tau)$**:** picks uniformly at random π_a and uses the trapdoor τ_b to run $\Pi_b.\mathsf{Simulate}(\tau_b, \pi_a, \mathcal{C})$ to get π_b. It outputs (π_a, π_b).

The correctness follows immediately from the correctness of the two proofs. Indeed if (π_a, π_b) is output by $\mathsf{no.Prove}(\mathcal{C}, w, s, \mathsf{crs}; \rho)$, π_a output should not verify (*i.e.* $\Pi_a.\mathsf{Verify}(\mathcal{C}, \pi_a) = 0$) as the input $(C, (w, s)) \notin \mathcal{R}$.

Theorem 1. *The proof system Π is sound if Π_a is correct and Π_b is sound.*

Proof. We assume there exists an adversary against the soundness of our proof system, we will show there exists an adversary $\mathcal{B}$ that can use this adversary to break the soundness of Π_b. An adversary against the soundness of our scheme outputs $(\mathcal{C}, \pi_a, \pi_b)$ such that while $\mathsf{no.Verify}(\mathcal{C}, \pi_a, \pi_b)$ holds, the plaintext in $\mathcal{C}$ does indeed belong to $\mathcal{L}$.

We will distinguish two kind of adversaries, those who compute π_a honestly, and those who does not. This means that either π_a is invalid for a valid word w and some randomness ρ, or that the adversary gives an incorrect output π_a and managed to build a proof π_b stating that it is computed correctly.

The adversary $\mathcal{B}$ now decrypts the ciphertext $\mathcal{C}$ to recover the word w, picks some fresh r' and encrypts it into a $\mathcal{C}'$ (this is a randomization of the first ciphertext) and tries to compute $\Pi_a.\mathsf{Prove}(\mathcal{C}', w, r'; \rho')$. Either with advantage greater than $\varepsilon/2$ this proof does not hold, and so $\mathcal{B}$ managed to break the correctness of the proof system Π_a. Either with advantage greater than $\varepsilon/2$ this proof holds, this means that the π_a output by the adversary was not correctly computed, hence this lead to breaking the soundness of Π_b. (As the second proof given by the adversary is valid while proving an incorrect statement). □

Theorem 2. *The proof system Π is zero-knowledge assuming the zero-knowledge property of Π_b and the indistinguishability of proofs of Π_a.*

Proof. It is easy to see that the output of $\mathsf{no.TSetup}$ and $\mathsf{no.Setup}$ are indistinguishable (by the zero-knowledge property of Π_b). We assume there exists an adversary against the Zero-Knowledge property of our scheme with advantage ε, we are going to follow a sequence of games to give an upper bound on this value. We have to prove that the distributions $\{\mathsf{Prove}(\mathsf{crs}, \mathcal{C}, (w, r))\}$ and $\{\mathsf{Simulate}(\mathsf{crs}, \mathcal{C})\}$ are indistinguishable for a ciphertext $\mathcal{C} = \mathsf{Encrypt}(\mathsf{ek}, w; r)$ with $w \notin \mathcal{L}$ generated by the adversary.

[3] This property does not hold for Groth Sahai proofs, given a correctly computed invalid proof for an equation like $\mathcal{X} = \mathcal{A}$, when in fact $\mathcal{X} = \mathcal{B}$, one can simply rely on the homomorphic properties of the verification to check if $GS.\mathsf{Verify}(\pi, \mathcal{C}, \mathcal{B})$ holds, hence distinguishing π from a random value.

$\mathcal{G}_0$ We start from the real game (*i.e.* $\pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, \mathcal{C}, (w, r))$).

$\mathcal{G}_1$ In this game, the simulator uses the Zero-Knowledge trapdoor τ_b from $\mathit{\Pi}_b$ to simulate the proofs on the valid statement π_a (*i.e.* it outputs $\pi_a := \mathit{\Pi}_a.\mathsf{Prove}(\mathcal{C}, (w, r); \rho)$ and $\pi_b := \mathit{\Pi}_b.\mathsf{Simulate}(\pi_a, \mathcal{C}, (w, r); \rho')$.
Here the adversary has advantage $\varepsilon_1 \leq \varepsilon + \mathsf{Adv}_{\mathsf{ZK}}$,

$\mathcal{G}_2$ In this game, the simulator outputs a random value instead of π_a and simulates π_b. Under the indistinguishability of the proofs on π_a, this game is similar to the previous one.
Here the adversary has advantage $\varepsilon_2 \leq \varepsilon_1 + \mathsf{Adv}_{\mathsf{ind}_\pi}$,

In this last game, one obtains the algorithm no.$\mathsf{Simulate}(\mathcal{C}, \tau)$ and we get $\varepsilon \leq \mathsf{Adv}_{\mathsf{ZK}} + \mathsf{Adv}_{\mathsf{ind}}$. □

3.2 Concrete Examples

In order to instantiate the previous proposition, we need some techniques to prove that a statement is invalid. To do so, we propose the approach consisting in generating a proof as if the statement was valid, and show that while this proof does not hold it was honestly generated.

As we show in our application to zero-knowledge with witness elimination in Section 4, Kiayias and Zhou aimed to do so in [KZ09], but incompletely. Additionally, they did it using an external proof system, that adds several rounds of interaction. (*i.e.* they use a sigma protocol to prove the (partial-)validity of a proof based on smooth projective hash functions.)

In the following we propose new techniques to do so via classical proof systems, first by proving the validity of a SPHF-proof via another SPHF, and then by mixing Groth-Sahai methodology with SPHF. Our generic approach from Section 3.1, requires the second proof to be zero-knowledge, and the first one to be homomorphicly randomizable and to achieve the indistinguishability on the proof. Groth Sahai provides the zero-knowledge while smooth projective hashing provides the indistinguishability via its pseudo-randomness.

We are going to work on SPHF-friendly languages. Recent works have drastically increased the range of languages manageable with SPHF (*e.g.* [BBC$^+$13a,BBC$^+$13b]), to every kind of pairing product equations over graded rings, so this will not really limit applications of theses techniques in concrete protocols.

We assume that languages $\mathfrak{L}$ have additional parameters $\mathfrak{L}_p, \mathfrak{L}_u$, the first part is a public description of a specificity of the language needed to build a smooth projective hash function on it, while the other one is private and is needed for verification purpose. (In case of a revocation language $\mathfrak{L}$, $\mathfrak{L}_p$ is a commitment to the revocation list, while $\mathfrak{L}_u$ is the randomness from the commitment.)

Non-Zero-Knowledge Proofs, Using Smooth Projective Hash Functions. To show that a word committed into $\boldsymbol{v}$ is not in a language described by $\mathfrak{L}_p$, one ends up doing the following (cf Figure 1):

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
V, \boldsymbol{v}, w_v & & \mathfrak{L}_p, \mathfrak{L}_u \\
\hline
(\mathsf{hp}_v, \mathsf{hk}_v) := (\mathit{\Gamma}_v(\mathfrak{L}_p) \odot \boldsymbol{\alpha_v}, \boldsymbol{\alpha_v}) & & \\
H_v := \mathit{\Theta}_v(V, \mathfrak{L}_p) \odot \boldsymbol{\alpha_v} & \xrightarrow{\mathfrak{V} = (\mathsf{hp}_v, H_v, \boldsymbol{v})} & H'_v := \mathfrak{L}_u \odot \mathsf{hp}_v \overset{?}{\neq} H_v \\
 & \xleftarrow{\mathsf{hp}_u} & (\mathsf{hp}_u, \mathsf{hk}_u) := (\mathit{\Gamma}_u(\mathfrak{V}) \odot \boldsymbol{\alpha_u}, \boldsymbol{\alpha_u}) \\
H'_u := (\boldsymbol{w_v}, \boldsymbol{\mathsf{hk}_v}) \odot \mathsf{hp}_u & \xrightarrow{H'_u} & H'_u \overset{?}{=} \mathit{\Theta}_u(\mathfrak{V}) \odot \mathsf{hk}_u.
\end{array}$$

Fig. 1. Generic SPHF-based proof of exclusion

Where the first SPHF is on the language described by $\mathfrak{L}_p$ while the other one is based on the language of a correct computation between a hash value, a projection key and a ciphertext (as there is a dependency between those two terms it seems improbable to be able to do better in this case).

A more concrete example. In order to explain the previous formalism, let us now give a more concrete example, with a language described by an ElGamal encryption of a word U. The prover possesses a word V, the verifier the word U and publishes an ElGamal ciphertext of U: $\mathfrak{L}_p = (h^s U, g^s)$. The prover encrypts his word V using ElGamal encryption scheme and proves to the verifier that V is not the plaintext encrypted in $\mathfrak{L}_p$. Following the previous technique we can achieve a 3-round proof as described on Figure 2. The second SPHF is smooth if and only if $V = U$, this means that technically an adversary can break the soundness of the verification of the valid computation of hp_v, H_v when the word V is different from U. However in this case, the protocol should already return yes so he cannot gain anything from doing so. The proof requires overall 10 group elements: 2 for the initial commit of U, 2 for the one of V, 2 overall for hp_v, H_v and 4 for hp_u, H'_u.

$$
\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
V, h^r V, g^r, r & & \mathfrak{L}_p = (h^s U, g^s), U, s \\
\hline
(\mathsf{hp}_v, \mathsf{hk}_v) := (h^\lambda g^\mu, (\lambda, \mu)) & \xrightarrow{\mathfrak{V}' = (\mathsf{hp}_v, h^r V, g^r)} & H'_v := \mathsf{hp}_v^s \\
 & \xleftarrow{\mathsf{hp}_u, \mathfrak{L}_p} & \mathsf{hp}_u := (h^\delta (h^s U/(h^r V))^\beta, g^\delta (g^s/g^r)^\beta, \mathsf{hp}_v^\beta g^\gamma), \\
 & & \mathsf{hk}_u := (\delta, \beta, \gamma) \\
H'_u := \mathsf{hp}_{u,1}^\lambda \mathsf{hp}_{u,2}^\mu \mathsf{hp}_{u,3}^r & \xrightarrow{H'_u, H_v} & H'_v \overset{?}{\neq} H_v \wedge H'_u \overset{?}{=} \mathsf{hp}_v^\delta H_v^\beta (g^r)^\gamma . \\
H_v := (h^s U/V)^\lambda (g^s)^\mu & &
\end{array}
$$

Fig. 2. Tweaked ElGamal based SPHF proof of inequality

We stress that, this construction differs from the generic approach in the sense that $\mathfrak{L}_p$ instead of being known before the protocol like in the generic construction from Figure 1, can be set on the fly and postpone to the second flow. While this proof is not zero-knowledge in any way, it will find some use in our LAKE (*Language-Authenticated Key Exchange*) application in Appendix C.2.

Zero-Knowledge Proofs, Using Groth-Sahai Non-Interactive Proof Technique.

We now want to supersede the last proof with a zero-knowledge proof. So once again we do a smooth projective hash function for the first language, and then prove using a Groth Sahai proof that we indeed know the associated hash key, such that the hash value and the projection key are consistent, that way we can reduce the protocol interactivity to one flow as explained in Figure 3.

$$
\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
V, \boldsymbol{v}, w_v & & \mathfrak{L}_p, \mathfrak{L}_u \\
\hline
(\mathsf{hp}_v, \mathsf{hk}_v) := (\mathit{\Gamma}_v(\mathfrak{L}_p) \odot \boldsymbol{\alpha_v}, \boldsymbol{\alpha_v}) & & \\
H_v := \mathit{\Theta}_v(V, \mathfrak{L}_p) \odot \boldsymbol{\alpha_v} & & \\
\pi = GS.\mathsf{Prove}(H_v \wedge \mathsf{hp}_v; \mathsf{hk}_w) & \xrightarrow{\mathfrak{V}' = \mathsf{hp}_v, \boldsymbol{v}, \pi, H_v} & H'_v := \mathfrak{L}_u \odot \mathsf{hp}_v \\
 & & H_v \overset{?}{\neq} H'_v \wedge \mathsf{Verify}(\pi).
\end{array}
$$

Fig. 3. Generic SPHF + NIZK proof of inequality

A more concrete example. If we consider our former example with a Groth Sahai proof instead of the second SPHF we end up with a one-round protocol. Overall this would require 4 elements in $\mathbb{G}_1$ for the hash proof, 6 other in $\mathbb{G}_1$ for the additional commitments, 4 in each group for the quadratic proof, 1 in $\mathbb{G}_1$ for one of the multi scalar exponentiation equation and 2 scalars for the other. Overall, the protocol is very efficient and only requires the transmission of 15 elements in $\mathbb{G}_1$, 4 elements in $\mathbb{G}_2$ and 2 scalars in $\mathbb{Z}_p$.

3.3 Transformation from a NIZK to a SS-NIZK

In one of the following applications, we will need our proof of exclusion to be simulation-sound ([Sah99]). There is a generic transformation from Groth Sahai based NIZK proofs to Simulation-Sound.

To construct a Simulation-Sound proof that some word w does not belong to a language $\mathfrak{L}$, one uses the following roadmap, assuming the common reference string crs contains a common reference string for the Groth-Sahai proof system crs_{GS}, a verification key pk for a Structure-Preserving Signature scheme [AFG$^+$10], and the prover already possesses a pair of primary keys $(\mathsf{psk}, \mathsf{ppk})$ for a one-time two-tier signature scheme [BS07][4]:

1. generates a secondary signing/verification key pair $(\mathsf{ssk}, \mathsf{spk})$ for the one-time two-tier signature.
2. commits to a random tuple of elements R corresponding to a signature.
3. generates, using Groth Sahai and our exclusion proof, a proof π that either w does not belong to $\mathfrak{L}$, or that R is a valid signature of the verification key spk of the one-time signature, under the public key pk contained in the CRS crs.
4. sends this proof π, the verification key of the one-time signature, and the corresponding one-time signature of everything under $(\mathsf{psk}, \mathsf{ssk})$.

Referring to [HJ12], it can be shown that this scheme is Zero-Knowledge under the Indistinguishability of the two types of Groth-Sahai common reference strings, and that both the simulation-soundness and the soundness come from the unforgeability of the two involved signatures.

4 Application to Zero-Knowledge with Witness Elimination

In [KZ09], Kiayias and Zhou introduced the notion of zero-knowledge proofs with witness elimination. They described it through a universally composable ideal functionality, directly giving strong guideline as how to achieve a generic construction.

Ideal Functionality for Zero-Knowledge with Witness Elimination In the universal composability (UC) framework, once a protocol is proved secure, it can be used in arbitrary contexts retaining its security properties (*i.e.* when composed with other instances of the same or other protocols). The security in this framework is defined in the sense of protocol emulation (*i.e.* a protocol P emulates some protocol P', if P does not affect the security of anything else than P' would have). To prove security in the UC framework, we define an ideal functionality $\mathcal{F}$ which can be thought of as an incorruptible trusted party that takes inputs from all parties and hands back outputs to the parties. The functionality $\mathcal{F}$ is a formal specification of a cryptographic task and is secure by definition. Hence, if a protocol P emulates $\mathcal{F}$, one can infer that it securely realizes the given task in arbitrary contexts.

The ideal functionality for Zero-Knowledge with Witness Elimination $\mathcal{F}_{ZKWE}^{\mathcal{R},\mathcal{Q}}$ proposed by Kiayias and Zhou builds upon that of Zero-Knowledge (proposed in 2001 by Canetti [Can01]) , with the extra requirement that the prover shows that it did not use some eliminated witnesses to prove the statement. We recall it on Figure 4.

It is parametrized by two binary relations $\mathcal{R}$ and $\mathcal{Q}$. The prover $\mathcal{P}$ is given an input $\langle x, w\rangle$ (Prove query) and the verifier $\mathcal{V}$ is given an input w' (Verify query). The verifier should only accept if $\mathcal{R}(x, w)$ and $\neg\mathcal{Q}(w, w')$ hold. Furthermore, it should not learn anything from the protocol except the value x and the existence of such a witness w. Following the authors of [KZ09], we do not deal here with adaptive corruptions. The functionality proceeds in three steps:

- Upon receiving a $(\mathsf{Prove}, \mathsf{sid}, \langle x, w\rangle)$ query from the prover $\mathcal{P}$: While blocking the secret inputs from the adversary, it leaks a bit through the $(\mathsf{LeakProve}, \mathsf{sid}, \langle x, \varphi\rangle, \mathcal{P})$ answer to tell the adversary whether $\mathcal{R}(x, w)$ holds. As explained in [KZ09], this does not affect the security properties of the protocol as long as the elimination relation

[4] This can easily be achieved by applying a Chameleon Hash on itself for example.

$\mathcal{Q}$ is such that the $\mathcal{F}_{ZKWE}^{\mathcal{R},\mathcal{Q}}$ functionality emulates the $\mathcal{F}_{ZK}^{\mathcal{R}}$ functionality (the prover can easily learn whether the witness is valid or not). The requirement on $\mathcal{Q}$ is that given a witness w, one can sample a witness w' such that $\mathcal{Q}(w, w')$ happens with negligible probability. This is in particular the case for the substring equality considered in their article and for the more general membership to the languages considered here (see Section 5).

- Upon receiving a $(\mathsf{Verify}, \mathsf{sid}, w')$ query from party $\mathcal{V}$: While blocking the secret inputs from the adversary, it leaks the information that $\mathcal{V}$ sent his witness through the $(\mathsf{LeakVerify}, \mathsf{sid}, \mathcal{V})$ answer.
- Upon receiving a $(\mathsf{InflVerify}, \mathsf{sid})$ query from the adversary: it leaks a last bit through the $(\mathsf{RetVerify}, \mathsf{sid}, b)$ answer to tell the adversary whether $\mathcal{Q}(w, w')$ holds. The query $\mathsf{InflVerify}$ can be asked only once, capturing a similar property to the resistance to offline dictionary attacks in the case of password-based protocols.

$\mathcal{F}_{ZKWE}^{\mathcal{R},\mathcal{Q}}$ is parametrized with the ZK relation $\mathcal{R}$ and the elimination relation Q.

- Upon receiving $(\mathsf{Prove}, \mathsf{sid}, \langle x, w\rangle)$ from party $\mathcal{P}$ where $\mathsf{sid} = (\mathcal{P}, \mathcal{V}, \mathsf{sid}')$, record $\langle \mathcal{P}, x, w\rangle$, send $(\mathsf{LeakProve}, \mathsf{sid}, \langle x, \varphi\rangle, \mathcal{P})$ to the adversary, where $\varphi = 1$ if $\mathcal{R}(x, w)$ holds, and $\varphi = 0$ otherwise. Ignore future $(\mathsf{Prove}, \ldots)$ inputs.
- Upon receiving $(\mathsf{Verify}, \mathsf{sid}, w')$ from party $\mathcal{V}$ where $\mathsf{sid} = (\mathcal{P}, \mathcal{V}, \mathsf{sid}')$, record $\langle T, w'\rangle$, send $(\mathsf{LeakVerify}, \mathsf{sid}, \mathcal{V})$ to the adversary. Ignore future $(\mathsf{Verify}, \ldots)$ inputs.
- Upon receiving $(\mathsf{InflVerify}, \mathsf{sid})$ from the adversary, if $\mathcal{R}(x, w)$ and $\neg\mathcal{Q}(w, w')$ hold, then send $(\mathsf{RetVerify}, \mathsf{sid}, 1)$ to party $\mathcal{V}$. Else if $\neg\mathcal{R}(x, w)$ holds or $\mathcal{Q}(w, w')$ holds, then send $(\mathsf{RetVerify}, \mathsf{sid}, 0)$ to party $\mathcal{V}$.

Fig. 4. Functionality $\mathcal{F}_{ZKWE}^{\mathcal{R},\mathcal{Q}}$

Generic Approach. After receiving the witness w' eliminated by the verifier, the construction of a zero-knowledge proof with witness elimination as presented by Kiayias and Zhou in [KZ09] requires two main parts by the prover. First, in a regular zero-knowledge proof, he starts by proving that the statement $\mathcal{R}(x, w)$ is indeed fulfilled, and then, in another part a little bit trickier, he has to prove that the witness he used does not belong to the elimination list, which is $\neg\mathcal{Q}(w, w')$. We recall on Figure 5 their original generic construction. In a nutshell, they proceed in three steps:

- In a first step, the verifier sends an encryption $\mathcal{C}'$ of the eliminated witness w' to the prover.
- In a second step, the prover computes an encryption $\mathcal{C}$ of his witness w, generates a hash key $\mathsf{hk} = \mathsf{HashKG}$, computes a projection key $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk})$ and computes the hash value $\kappa = \mathsf{Hash}(\mathcal{C}', (\mathcal{C}, w), \mathsf{hk})$. He sends hp and κ to the verifier. The aim of this smooth projective hash function is to ensure that $w \neq w'$.
- In a last step, both players engage in a zero-knowledge proof of membership (ZKPM) subprotocol to show that $\mathcal{R}(x, w)$ holds.

A Flaw in the Original Approach. In the protocol presented in [KZ09] that we recalled in the previous section, Kiayias and Zhou propose to prove that $\neg\mathcal{Q}(w, w')$ by doing an implicit proof of equality (*i.e.* $\mathcal{Q}(w, w')$) using a smooth projective hash function. They make the prover send the hash value κ, and ask the verifier to check whether it is equal to the projected hash value κ'. If those values are different, then the relation $\mathcal{Q}(w, w')$ does not stand, which proves that the witness is not eliminated.

However, this requires the prover to be honest in this process, as if he sends a inconsistent projection key to the verifier, then it will lead to an inequality between the hash and the projected hash values, meaning that the proof of membership to the elimination list (*i.e.* $\mathcal{Q}(w, w')$) will not hold, and finally that he will be able to convince the verifier with an invalid statement with overwhelming probability.

Common reference string: $\mathsf{crs} = (\mathsf{pk}, \rho)$, where pk is a public key of an encryption scheme $\mathcal{E}$, and ρ is a reference string of a Zero-Knowledge Proof of Membership (ZKPM) scheme.

Protocol steps: Upon receiving $(\mathsf{Verify}, \mathsf{sid}, w')$ from the environment, party $\mathcal{V}$ selects $r' \overset{\$}{\leftarrow} U$ and computes $\mathcal{C}' = \mathsf{Encrypt}(\mathsf{pk}, w'; r')$, and sends $(\mathsf{move}_1, \mathcal{C}')$ to party $\mathcal{P}$.
Upon receiving $(\mathsf{Prove}, \mathsf{sid}, x, w)$ from the environment, party $\mathcal{P}$ first checks if $(x, w) \in \mathcal{R}$ and waits for a move_1 message from party $\mathcal{V}$. After receiving move_1 message:

- if $\neg\mathcal{R}(x, w)$ holds, then party $\mathcal{P}$ sends party $\mathcal{V}$ a message $(\mathsf{move}_2, \text{“no valid proof”})$,
- else if $\mathcal{R}(x, w)$ holds, then party $\mathcal{P}$ computes $\mathcal{C} = \mathsf{Encrypt}(\mathsf{pk}, w; r)$, and then party $\mathcal{P}$ sends party $\mathcal{V}$ a message $(\mathsf{move}_2, \mathsf{hp}, \kappa)$, where $\mathsf{hk} = \mathsf{HashKG}$, $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk})$, $\kappa = \mathsf{Hash}(\mathcal{C}', (\mathcal{C}, w), \mathsf{hk})$. Now, parties $\mathcal{P}$ and $\mathcal{V}$ play the roles of prover and verifier respectively to run a ZKPM subprotocol, to show that $x, \mathcal{C}', \kappa$ is consistent:

$$\exists(w, r), (x, w) \in R \wedge \mathcal{C} = \mathsf{Encrypt}(\mathsf{pk}, w; r) \wedge \kappa = \mathsf{Hash}(\mathcal{C}', (\mathcal{C}, w), \mathsf{hk}).$$

Upon receiving $(\mathsf{move}_2, \text{“no valid proof”})$ from party $\mathcal{P}$, party $\mathcal{V}$ returns $(\mathsf{RetVerify}, \mathsf{sid}, 0)$ to the environment. Else if receiving $(\mathsf{move}_2, \mathsf{hp}, \kappa)$, party $\mathcal{V}$ computes $\kappa' = \mathsf{ProjHash}(\mathcal{C}, (\mathcal{C}', w', r'), \mathsf{hp})$ and if $\kappa \neq \kappa'$ party and $\mathcal{V}$ accepts the ZKPM proof in the subprotocol above, then party $\mathcal{V}$ returns $(\mathsf{RetVerify}, \mathsf{sid}, 1)$ to the environment; otherwise returns $(\mathsf{RetVerify}, \mathsf{sid}, 0)$ to the environment.

Fig. 5. Initial Generic construction of the protocol for Zero-Knowledge Proofs with Witness Elimination

This issue comes from the fact that the validity of the projection key hp is nowhere verified in the generic description. Intuitively, this verification should be part of the following ZKPM subprotocol. In their description, the latter does not involve the computation of hp in any way, so that there is the possibility for the prover to send a bogus one in order to avoid collision for words in the elimination list. This way, the prover is able to get its ZKPM proof for $\mathcal{R}(x, w)$ accepted while using a witness of the elimination list, without being caught by the verifier.

5 A Generic Fix and Several Concrete Instantiations

Improvement. Before fixing Kiayias and Zhou's protocol, we start by giving some other improvements. First, the authors only considered equalities or substring equalities for the relation $\mathcal{Q}$. We improve this by allowing $\mathcal{Q}$ to be a more general relation of membership to a language specified by w': $\mathcal{Q}(w, w') \Leftrightarrow w \in \mathcal{L}_{w'}$. More precisely, following 3.1, we assume that the description of the language is public, meaning that given w', one learns $\mathcal{L}_{w'}$ automatically. We also assume that given w, one can easily and publicly check whether $w \in \mathcal{L}_{w'}$ or $w \notin \mathcal{L}_{w'}$.

Furthermore, in order to be able to satisfy $\mathcal{F}_{ZKWE}^{\mathcal{R},\mathcal{Q}}$, we assume that given $\mathcal{L}_{w'}$, the simulator will be able to generate $w_1 \in \mathcal{L}_{w'}$ and $w_2 \notin \mathcal{L}_{w'}$. This is a natural assumption and can be achieved in different ways: either, this is publicly achievable by anybody; Or the language $\mathcal{L}_{w'}$ is randomizable and w' includes a witness included in $\mathcal{L}_{w'}$ from which it is possible to generate $w \in \mathcal{L}_{w'}$; Or we assume that $\mathcal{S}$ possesses a trapdoor (stored into the CRS). In most applications, we will be in the second case, where the language $\mathcal{L}_{w'}$ is randomizable.

Our second improvement is to replace most of the interactive proofs by possibly *non-interactive* proofs of knowledge, using the proofs of non-membership given in Section 3. More details follow in the next section.

A Generic Fix. As already explained in Section 4, the problem of the initial generic protocol given by the authors of [KZ09] lies in the Sigma protocol, which does not involve the computation of hp in any way. To avoid this issue, we now include in the Zero-Knowledge proof a new proof that the projection key hp was correctly generated.

Our new generic protocol is presented on Figure 6. From a high point of view, the proof π_1 is the same as in [KZ09] and proves the validity of the statement x under the witness w, namely that $\mathcal{R}(x, w)$ holds.

The non-interactive proof π_2 replaces their interactive ZKPM subprotocol, and ensures that $\mathcal{Q}(w, w')$ does not hold (*i.e.* w is not in the exclusion language). As explained in Section 3, this proof consists in the combination of a ciphertext $\mathcal{C}$ of w and a proof π_2 of non-membership. This latter proof is divided into a proof π_a of membership of w in the language defined by w' (which should not hold), and a second proof π_b showing that, while the proof π_a does not hold, it has been honestly generated, so there exists some randomness used to prove π_1, and some extra randomness, such that the expected proof of equality is indeed π_a. By completeness of the proof, this proves that w is indeed not included in the language defined by w'.

Common Reference String: $\mathsf{crs} = (\mathsf{pk}, \rho_1, \rho_2)$, where pk is a public key of an encryption scheme $\mathcal{E}$ (both $\mathsf{Encrypt}$ and $\mathsf{no.Encrypt}$ use $\mathcal{E}$), ρ_1 is a reference string of a zero-knowledge proof of membership (for the relation $\mathcal{R}$), and ρ_2 is a reference string of a zero-knowledge proof of non-membership (for the relation $\mathcal{Q}$).

Protocol steps: Upon receiving $(\mathsf{Verify}, \mathsf{sid}, w')$ from the environment, party $\mathcal{V}$ selects a random r' and computes $\mathcal{C}' = \mathsf{Encrypt}(\mathsf{pk}, w'; r')$, and sends $(\mathsf{move}_1, \mathcal{C}')$ to party $\mathcal{P}$.

Upon receiving $(\mathsf{Prove}, \mathsf{sid}, x, w)$ from the environment, party $\mathcal{P}$ first checks if $\mathcal{R}(x, w)$ holds and waits for a move_1 message from party $\mathcal{V}$. After receiving move_1 message:

- if $\neg\mathcal{R}(x, w)$ holds, then party $\mathcal{P}$ sends party $\mathcal{V}$ a message $(\mathsf{move}_2, \text{“no valid proof”})$,
- else if $\mathcal{R}(x, w)$ holds, then party $\mathcal{P}$ selects three random values r, $\mathbf{r_1}$ and $\mathbf{r_2}$ and sends party $\mathcal{V}$ a message $(\mathsf{move}_2, \pi_1, (\mathcal{C}, \pi_2))$, where $\pi_1 = \mathsf{ZK.Prove}(x, w, \mathcal{R}; \mathbf{r_1})$ and $(\mathcal{C}, \pi_2) = \mathsf{SS.no.Prove}(w, w', \mathcal{Q}, \mathbf{r_1}; \mathbf{r_2})$, meaning in particular that $\mathcal{C} = \mathsf{Encrypt}(\mathsf{pk}, w; r)$.

Upon receiving $(\mathsf{move}_2, \text{“no valid proof”})$ from party $\mathcal{P}$, party $\mathcal{V}$ returns $(\mathsf{RetVerify}, \mathsf{sid}, 0)$ to the environment. Else if receiving $(\mathsf{move}_2, \pi_1, (\mathcal{C}, \pi_2))$, party $\mathcal{V}$ checks both proofs and returns $(\mathsf{RetVerify}, \mathsf{sid}, 1)$ to the environment if they are correct, and $(\mathsf{RetVerify}, \mathsf{sid}, 0)$ otherwise.

Fig. 6. Generic Construction of Zero Knowledge proof with Witness Elimination

Theorem 3. *This generic construction fulfils the Zero-Knowledge with Witness-Elimination Functionality $\mathcal{F}_{ZKWE}^{\mathcal{R},\mathcal{Q}}$ under the assumption that π_1 is a zero-knowledge proof of membership for the relation $\mathcal{R}$, and π_2 is a zero-knowledge proof of non-membership for the relation $\mathcal{Q}$, as defined in section 3.*

Due to lack of space, the proof of Theorem 3 is provided in Appendix B =

Concrete Instantiation of the Fixed and Improved Protocol. In their original article [KZ09], Kiayias and Zhou present a concrete instantiation of the protocol, where a user proves being in a possession of a valid pair (m, σ) of a Boneh Boyen [BB04] signature σ, on a message m. To show the potency of our approach, we propose to instantiate their original scheme more efficiently, in a round-optimal way, without the initial flaw described in Section 4. In order to have an easy message recovery, we are going to do a naive bit per bit commitment. While this is not necessarily the most efficient approach in practice, asymptotically this is already more efficient than their use of Paillier encryption (a quadratic cost instead of a cubic one using Paillier encryption).

The scheme is described on Figure 7. The proof consists of a commitment to σ, λ, μ, θ (*i.e.* 12 group elements), and a proof of two linear multi-scalar exponentiation equations (2 elements each), one of a quadratic (9 group elements), and ℓ quadratic scalar equations. Overall, $31 + 9\ell$ group elements are exchanged [5] in two flows. For a concrete security parameter, the initial (flawed) scheme was more efficient but required 2 additional rounds. However, as many elements live in a RSA modulus space, asymptotically our scheme has a better efficiency, both in number of rounds and communication size.

[5] This estimation is very rough, and optimization like running a KDF on the hash value could further improve the efficiency, but that is beyond the point of this construction.

$$
\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
(m,\sigma) & & \sigma' \\
\hline
 & & r', s' \xleftarrow{\$} \mathbb{Z}_p \\
C_v := \mathsf{Encrypt}(\sigma; t, z) & \xleftarrow{\quad c' \quad} & c' := \mathsf{Encrypt}(\sigma'; r', s') \\
\mathsf{hp}_v := (u^\lambda g^\mu, v^\theta g^\mu) & & \\
\mathsf{hk}_v := (\lambda, \theta, \mu) & & \\
H_v := (c'_3/\sigma)^\mu (c'_1)^\lambda (c'_2)^\theta & & \\
\pi = GS.\mathsf{Prove}(\mathsf{hp}_v, H_v; \mathsf{hk}_v, C_v; & \xrightarrow{(\mathsf{hp}_v, C_v, \pi)} & H'_v := \mathsf{hp}_v^{(r',s')} = (u^\lambda g^\mu)^{r'} (v^\theta g^\mu)^{s'} \\
\sigma, t, z, \mathsf{hk}_v) & & H'_v \overset{?}{\neq} H_v \wedge \mathsf{Verify}(\pi).
\end{array}
$$

The proof π considers the language:

$$
\exists t, z, m, \lambda, \theta, \mu, \sigma, \begin{cases} C_v = \mathsf{Encrypt}(\sigma; t, z) \\ \mathsf{hp}_v = (u^\lambda g^\mu, v^\theta g^\mu) \\ H_s = (c'_3/\sigma)^\mu (c'_1)^\lambda (c'_2)^\theta \\ e(\sigma, \mathsf{vk} g^m) = 1_T \end{cases}
$$

Fig. 7. Concrete Construction of zero-knowledge with witness elimination

References

ACHdM05. Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. `http://eprint.iacr.org/2005/385`.

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

AFG$^+$10. Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 209–236. Springer, August 2010.

BB04. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer, May 2004.

BBC$^+$13a. Fabrice Ben Hamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. Efficient UC-secure authenticated key-exchange for algebraic languages. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 272–291. Springer, February / March 2013.

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

BCKL08. Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 356–374. Springer, March 2008.

BG13. Stephanie Bayer and Jens Groth. Zero-knowledge argument for polynomial evaluation with application to blacklists. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 646–663. Springer, May 2013.

BNF13. Nasima Begum, Toru Nakanishi, and Nobuo Funabiki. Efficient proofs for CNF formulas on attributes in pairing-based anonymous credential system. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology - ICISC 2012*, volume 7839 of *Lecture Notes in Computer Science*, pages 495–509. Springer Berlin Heidelberg, 2013.

Bon98. Dan Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *Lecture Notes in Computer Science*. Springer, 1998. Invited paper.

BPV12. Olivier Blazy, David Pointcheval, and Damien Vergnaud. Round-optimal privacy-preserving protocols with smooth projective hash functions. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 94–111. Springer, March 2012.

BS07. Mihir Bellare and Sarah Shoup. Two-tier signatures, strongly unforgeable signatures, and Fiat-Shamir without random oracles. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 201–216. Springer, April 2007.

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

CCs08. Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 234–252. Springer, December 2008.

CG08. Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08: 15th Conference on Computer and Communications Security*, pages 345–356. ACM Press, October 2008.

CGH04. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

Cha85. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.

CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer, May 2001.

CLOS02. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, pages 494–503. ACM Press, May 2002.

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

Dam88. Ivan Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 328–335. Springer, August 1988.

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

GM84. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

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

HJ12. Dennis Hofheinz and Tibor Jager. Tightly secure signatures and public-key encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 590–607. Springer, August 2012.

ILV11. Malika Izabachène, Benoît Libert, and Damien Vergnaud. Block-wise p-signatures and non-interactive anonymous credentials with efficient attributes. In Liqun Chen, editor, *IMA Int. Conf.*, volume 7089 of *Lecture Notes in Computer Science*, pages 431–450. Springer, 2011.

Kal05. Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 78–95. Springer, May 2005.

KZ09. Aggelos Kiayias and Hong-Sheng Zhou. Zero-knowledge proofs with witness elimination. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 124–138. Springer, March 2009.

LRSW99. Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *SAC 1999: 6th Annual International Workshop on Selected Areas in Cryptography*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, August 1999.

Pen11. Kun Peng. A general, flexible and efficient proof of inclusion and exclusion. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 33–48. Springer, February 2011.

PP03. Duong Hieu Phan and David Pointcheval. Chosen-ciphertext security without redundancy. In Chi-Sung Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 1–18. Springer, November / December 2003.

Sah99. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science*, pages 543–553. IEEE Computer Society Press, October 1999.

A Zero-Knowledge Definitions

A.1 General definitions

Let R be an efficiently computable binary relation. For pairs $(x, w) \in R$, we call x the *statement* and w the witness. Let $\mathfrak{L}$ be the language consisting of statements in R.

Definition 4. *A zero-knowledge proof system for a language* $\mathfrak{L}$ *is a pair* $(\mathsf{Prove}, \mathsf{Verify})$ *of (interactive) Turing machines where* Verify *is probabilistic polynomial time, with the following three properties:*

- Completeness*: for all* $(x, w) \in R$*, on input* x*,* Verify *always outputs* 1 *when interacting with* Prove *with input* (x, w)
$$\langle \mathsf{Prove}(x, w) \leftrightarrow \mathsf{Verify}(x) \rangle \rightsquigarrow 1$$
*(*i.e. *if the statement is true, the honest verifier will be convinced of this fact by an honest prover).*
- Soundness*: for all* $x \notin \mathfrak{L}$*, for any (interactive) Turing machine* Prove^**, on input* x*,* Verify *outputs* 0 *with overwhelming probability when interacting with* Prove^* *with input* x
$$\langle \mathsf{Prove}^*(x) \leftrightarrow \mathsf{Verify}(x) \rangle \rightsquigarrow 0$$
*(*i.e. *if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability).*
- Zero-knowledge*: for any probabilistic polynomial time Turing machine* Verify^**, there exists a probabilistic polynomial time Turing machine* $\mathsf{Simulate}$ *such that for all* $(x, w) \in R$ *and all* $z \in \{0, 1\}^*$*, the transcript of the interaction of* Prove *and* Verify^* *on respective inputs* (x, w) *and* (x, z) *is indistinguishable from the output of* $\mathsf{Simulate}(x, z)$
$$\langle \mathsf{Prove}(x, w) \leftrightarrow \mathsf{Verify}^*(x, z) \rangle \approx \mathsf{Simulate}(x, z)$$
*(*i.e. *if the statement is true, no cheating verifier learns anything other than this fact).*

Definition 5. *A* non-interactive *zero-knowledge proof system for a language* $\mathfrak{L}$ *is a tuple*

$$(\mathsf{Setup}, \mathsf{TSetup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Simulate})$$

of probabilistic polynomial time Turing machines, with the following three properties:

- Completeness*: given* crs *output by* Setup*, a pair* $(x, w) \in R$*, and a proof* π *output by* $\mathsf{Prove}(\mathsf{crs}, x, w)$*,* Verify *always outputs* 1 *on input* (crs, x, π) (i.e. *if the statement is true, the honest verifier will be convinced of this fact by an honest prover).*
- Soundness*: given* crs *output by* Setup*, given* $x \notin \mathfrak{L}$*, for any bit-string* π^**,* Verify *outputs* 0 *with overwhelming probability on input* (crs, x, π^*) (i.e. *if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability).*
- Zero-knowledge*: for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *we have*
$$\Pr[\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^k) : \mathcal{A}(\mathsf{crs}) = 1] \approx \Pr[(\mathsf{crs}, \tau) \xleftarrow{\$} \mathsf{TSetup}(1^k) : \mathcal{A}(\mathsf{crs}) = 1]$$
and
$$\Pr[(\mathsf{crs}, \tau) \xleftarrow{\$} \mathsf{TSetup}(1^k), (x, w) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}), \pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, x, w) : \mathcal{A}(\pi) = 1]$$
$$\approx \Pr[(\mathsf{crs}, \tau) \xleftarrow{\$} \mathsf{TSetup}(1^k), (x, w) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}), \pi \xleftarrow{\$} \mathsf{Simulate}(\mathsf{crs}, x, \tau) : \mathcal{A}(\pi) = 1]$$

B Proof of Theorem 3

We prove the protocol by showing that, for any adversary $\mathcal{A}$, one can construct a simulator $\mathcal{S}$ such that, for every environment $\mathcal{Z}$, the interaction between, for one hand the environment, the players and the adversary (the real world), and for the other hand the environment, the ideal functionality and the simulator (the ideal world), are indistinguishable for the environment.

For the most part, the simulator simulates the protocol by executing it honestly on behalf of the honest parties, but using random dummy witnesses as inputs since the secret inputs (given to them by the environment) are unknown to the simulator. Furthermore, the simulator uses the simulators for the proof systems instead of the regular prove algorithms. More details follow. Recall that we only deal with static adversaries, which means that we know in advance which user is corrupted, and that the adversary cannot corrupt any player during an execution of the protocol.

Setup. The simulator generates the parameters, knowing the encryption scheme trapdoor (*i.e.* the decryption key) and the trapdoors for the reference strings of the proof schemes. It enables him to be able to use the simulation algorithms $\mathsf{ZK.Simulate}$ and $\mathsf{SS.no.Simulate}$ instead of the proving algorithms $\mathsf{ZK.Prove}$ and $\mathsf{SS.no.Prove}$.

When $\mathcal{P}$ and $\mathcal{V}$ are honest.
VERIFY STAGE: After receiving $(\mathsf{LeakVerify}, \mathsf{sid}, \mathcal{V})$ from the functionality, the simulator $\mathcal{S}$ chooses a random witness $\widetilde{w}'$, computes honestly the ciphertext $\mathcal{C}' = \mathsf{Encrypt}(\mathsf{pk}, \widetilde{w}'; r')$ and sends it to $\mathcal{P}$ in a move_1 message on behalf of $\mathcal{V}$.
PROVE STAGE: We consider two cases.

After receiving $(\mathsf{LeakProve}, \mathsf{sid}, \langle x, 0\rangle, \mathcal{P})$ from the functionality and any move_1 message from $\mathcal{V}$, then $\mathcal{S}$ sends $(\mathsf{move}_2, \text{“no valid proof”})$ to $\mathcal{V}$ on behalf of $\mathcal{P}$.

After receiving $(\mathsf{LeakProve}, \mathsf{sid}, \langle x, 1\rangle, \mathcal{P})$ from the functionality and a message $(\mathsf{move}_1, \mathcal{C}')$ from $\mathcal{V}$ containing a ciphertext $\mathcal{C}'$, the simulator uses a randomly chosen $\widetilde{w}$ to compute $\mathcal{C}$, and uses the simulation algorithms $\mathsf{ZK.Simulate}$ and $\mathsf{SS.no.Simulate}$ to compute π_1 and π_2. It then sends party $\mathcal{V}$ a message $(\mathsf{move}_2, \pi_1, (\mathcal{C}, \pi_2))$ on behalf of $\mathcal{P}$.
INFLUENCED VERIFY STAGE: After receiving the move_2 message from party P, $\mathcal{S}$ sends $(\mathsf{InflVerify}, \mathsf{sid})$ to the functionality.

This simulation is computationally indistinguishable from the real game due to the zero-knowledge property of the proof systems, and the indistinguishability property of the ciphertext.

When $\mathcal{P}$ is corrupted.
VERIFY STAGE: After receiving $(\mathsf{LeakVerify}, \mathsf{sid}, \mathcal{V})$ from the functionality, the simulator $\mathcal{S}$ chooses a random witness $\widetilde{w}'$, computes honestly a ciphertext $\mathcal{C}' = \mathsf{Encrypt}(\mathsf{pk}, \widetilde{w}'; r')$ and sends it to $\mathcal{P}$ in a $(\mathsf{move}_1, \mathcal{C}')$ message on behalf of $\mathcal{V}$.
PROVE STAGE: If $\mathcal{S}$ receives a $(\mathsf{move}_2, \text{“no valid proof”})$ message from the corrupted $\mathcal{P}$, the simulator sends $(\mathsf{Prove}, \mathsf{sid}, \langle x, \perp\rangle)$ to the functionality on behalf of $\mathcal{P}$, with $\neg\mathcal{R}(x, \perp)$.

If $\mathcal{S}$ receives a $(\mathsf{move}_2, \pi_1, (\mathcal{C}, \pi_2))$ message from a corrupted party $\mathcal{P}$, $\mathcal{S}$ executes honestly the verification protocols $\mathsf{ZK.Verify}$ and $\mathsf{SS.no.Verify}$ of the zero-knowledge proofs. If these verifications succeed, the simulator uses its trapdoor for the encryption scheme $\mathsf{no.Encrypt}$ to decrypt $\mathcal{C}$ and recover w. It then sends $(\mathsf{Prove}, \mathsf{sid}, \langle x, w\rangle)$ to the functionality on behalf of $\mathcal{P}$. If these verifications fail, it sends $(\mathsf{Prove}, \mathsf{sid}, \langle x, \perp\rangle)$ to the functionality on behalf of $\mathcal{P}$, with $\neg\mathcal{R}(x, \perp)$.
INFLUENCED VERIFY STAGE: After receiving a message $(\mathsf{LeakProve}, \mathsf{sid}, \langle x, \varphi\rangle, \mathcal{P})$ from the functionality, $\mathcal{S}$ sends $(\mathsf{InflVerify}, \mathsf{sid})$ to the functionality.

The perfect bindness of the encryption scheme ensures the correctness of the decryption and thus of the recovery of the witness w sent by the adversary.

If the adversary uses an input $\langle x, w\rangle$ such that $\mathcal{R}(x, w)$ and $\neg\mathcal{Q}(w, w')$ hold, then the correctness of the proof systems ensure that the verification made by the simulator will succeed.

On the contrary, if it uses an input such that at least one of the two relations does not hold, then the simulation-soundness of the proof systems ensure that the verification made by the simulator will fail. Recall that the simulator is likely to make false zero-knowledge proofs while simulating an honest prover, so that the adversary will most certainly have seen false proofs. This explains the need for simulation-soundness.

Due to these reasons, this simulation is thus computationally indistinguishable to the real world.

When $\mathcal{V}$ is corrupted.
VERIFY STAGE: After receiving $(\mathsf{LeakProve}, \mathsf{sid}, \langle x, \varphi\rangle, \mathcal{P})$ from the functionality and a $(\mathsf{move}_1, \mathcal{C}')$ message from the corrupted $\mathcal{V}$, then $\mathcal{S}$ uses its trapdoor for the encryption scheme $\mathsf{Encrypt}$ to decrypt $\mathcal{C}'$ and recover w'. It then sends $(\mathsf{Verify}, \mathsf{sid}, w')$ to the functionality on behalf of $\mathcal{V}$.

INFLUENCED VERIFY STAGE: After receiving $(\mathsf{LeakVerify}, \mathsf{sid}, \mathcal{V})$ from the functionality, the simulator $\mathcal{S}$ sends a query $(\mathsf{InflVerify}, \mathsf{sid})$ to the functionality.

PROVE STAGE: If $\varphi = 0$, then $\mathcal{S}$ sends $(\mathsf{move}_2, \text{“no valid proof”})$ to $\mathcal{V}$ on behalf of $\mathcal{P}$.

If $\varphi = 1$, we face two cases. If the functionality returns $(\mathsf{RetVerify}, \mathsf{sid}, 0)$, this means that $w \in \mathcal{L}_{w'}$. The simulator $\mathcal{S}$ then generates a witness $w \in \mathcal{L}_{w'}$ and uses it to generate honestly a message $(\mathsf{move}_2, \pi_1, (\mathcal{C}, \pi_2))$, using $\mathsf{ZK.Prove}$ and $\mathsf{SS.no.Prove}$, that it sends to $\mathcal{V}$ on behalf of $\mathcal{P}$. If the functionality returns a message $(\mathsf{RetVerify}, \mathsf{sid}, 1)$, this means that $w \notin \mathcal{L}_{w'}$. The simulator $\mathcal{S}$ then generates a witness $w \notin \mathcal{L}_{w'}$ and uses it to generate honestly a ciphertext $\mathcal{C} = \mathsf{no.Encrypt}(\mathsf{pk}, w; r)$. It then uses the simulation algorithms $\mathsf{ZK.Simulate}$ and $\mathsf{SS.no.Simulate}$ to compute π_1 and π_2 and sends party $\mathcal{V}$ a message $(\mathsf{move}_2, \pi_1, (\mathcal{C}, \pi_2))$ on behalf of $\mathcal{P}$.

The encryption scheme being perfectly binding ensures the correctness of the decryption and thus of the recovery of the witness w sent by the adversary. This simulation is then computationally indistinguishable from the real game due to the zero-knowledge property of the proof systems.

C Other Applications

C.1 Anonymous Credentials not Verifying a Property

Anonymous Credentials were introduced by Chaum in [Cha85], and were widely used ever since as a means for users to authenticate themselves while protecting their privacy (see, e.g. [Dam88,LRSW99,CL01,BCKL08]).

Most constructions consist in a first interaction, where a user obtains a signature on some message which corresponds to “his credentials”. When this user wants to authenticate, he then proves that he knows a signature on a message he does not want to leak but that fulfils some property.

Camenisch and Groß[CG08] proposed a way to build anonymous credentials with efficient attributes, and more recently Izabachène et al. proposed in [ILV11] a nearly non-interactive instantiation of this protocol. Their protocol requires an interaction to prove the AND of several credentials. Interestingly, their technique for proving the NAND of credentials is non-interactive, so combining this technique with our proof of No-Statement we can non-interactively prove the AND of several credentials by doing a NOT(NAND). Similarly, we can transform their non-interactive NOR into the non-interactive OR they were lacking, rendering their protocol completely non-interactive. Another transformation to achieve non-interactivity was recently sketched in [BNF13], however it requires an accumulator and so induces stronger and non-standard hypotheses.

Another application of our technique allows to conceive non-interactive anonymous credentials where an individual may require information but at the time of the proof may not necessarily know if he possesses the appropriate credential (like for example if the server revokes some secret credential). Once again our proof technique leads to such instantiation.

C.2 Language Authenticated Key Exchange

In [BBC$^+$13a], the authors introduced the notion of Language Authenticated Key Exchange which allows two users to agree on a shared session key if and only if they each possess a word in a language chosen secretly by the other.

Interestingly, the construction requires their languages to be “randomizable” which fits well with our description. In [BBC$^+$13a], they handle AND connectives of languages (*i.e.* intersection of languages) and limited form of OR connectives (*i.e.* union of languages). However, there was no known way to handle exclusion languages, but our technique allows to solve this and so extend the range of advanced languages manageable by those LAKE. We can also remove the limitation on the union of languages by using the connective NOT(AND(NOT)).

In their LAKE constructions, the SPHF is of course not zero-knowledge, and no simulation is required on this part (as everything is managed by the UC commitment) so one can use our proofs without requiring the “simulation sound” part. An interesting trade-off can also be achieved if the proof of validity of the SPHF computation is managed with a SPHF instead of a Groth Sahai proof, at the cost of (at most) one extra round in the protocol which would allow to avoid the use of pairings.

We now present our 3-round No-Proof, in order to prove that a word x is not in a language $\mathfrak{L}$:

- $\mathsf{Prove}(x, \mathfrak{L})$: This algorithm is interactive between P_i and P_j:
  1. P_i initiates the protocol by computing $\mathcal{C} = \mathsf{Encrypt}(w; r)$ and $(\mathsf{hp}, \mathsf{hk}, H)_{\mathsf{no}}(\mathcal{C}, \mathfrak{L}, r)$. P_i sends $\mathcal{C}, \mathsf{hp}$ to P_j.
     P_j generates $\mathsf{hk}_j, \mathsf{hp}_j = \mathsf{SPHF.Gen}(\mathfrak{L}_2, H_{\mathsf{no}}, \mathsf{hp}_{\mathsf{no}})$, and hp_j.
     P_i computes $H' = \mathsf{SPHF.ProjHash}(\mathfrak{L}_2, \mathsf{hp}_j, \mathsf{hk}_{\mathsf{no}}, x)$ and sends H'.
  2. P_j accepts if and only if $H \neq \mathsf{SPHF.ProjHash}(\mathfrak{L}, \mathcal{C})$ and $H' = \mathsf{SPHF.Hash}(\mathfrak{L}_2, H_{\mathsf{no}}, \mathsf{hp}_{\mathsf{no}}, \mathsf{hk}_j)$.

In the inner part of a LAKE protocol, those proofs can be run simultaneously on both sides, making it 4 flows instead of 2 in the original paper, but it allows to handle previous languages as well as their complements.