Multi-Issuer Anonymous Credentials Without a Root Authority

Kaoutar Elkhiyaoui, Angelo De Caro, and Elli Androulaki

IBM Research - Zurich
{kao, adc, lli}@zurich.ibm.com

Abstract. The rise of blockchain technology has boosted interest in privacy-enhancing technologies, in particular, anonymous transaction authentication. Permissionless blockchains realize transaction anonymity through one-time pseudonyms, whereas permissioned blockchains leverage *anonymous credentials*. Earlier solutions of anonymous credentials assume a single issuer; as a result, these solutions hide the identity of users but still reveal the identity of the issuer. A countermeasure is *delegatable credentials*, which supports multiple issuers as long as a root authority exists. Assuming a root authority however, is unsuitable for blockchain technology and decentralized applications. This paper introduces a solution for anonymous credentials that guarantees *user anonymity*, even without a root authority. The proposed solution is secure in the *universal composability* framework and allows users to produce anonymous signatures that are *logarithmic* in the number of issuers and *constant* in the number of user attributes.

1 Introduction

With the advent of blockchain technology, demand for efficient and practical anonymous transaction authentication has surged. Blockchain applications require all participants to verify whether the author of a given transaction is authorized to update the state of the shared ledger or not. When this verification leaks the identity of the transaction origin, this puts at risk the privacy of blockchain users. Permissionless blockchains tackle this issue with *one-time pseudonyms*. Permissioned blockchains, on the other hand, are expected to reconcile transaction anonymity with *user accountability*, making one-time pseudonyms inadequate. In other words, users should be able to sign transactions with their long-term identities, but at the same time, preserve their anonymity.

Anonymous credentials [10, 8] are cryptographic primitives that grant users the ability to sign messages and prove statements about themselves, without revealing their long-term identities. As such, they can be used to authenticate blockchain transactions anonymously. For example, Hyperledger Fabric [1] uses IDemix [2] – an implementation of the solution described in [9], to ensure that only users with certified identities and certain properties can submit transactions. Central to anonymous credentials is a trusted issuer that produces credentials for users. A credential is a signature from the issuer on a secret key and a set of attributes describing user properties. To sign messages anonymously, users leverage zero-knowledge proofs to demonstrate that they have received a valid credential from the issuer and that they know the corresponding secret key. These proofs can be extended so that users prove statements about themselves while signing messages.

Although these techniques protect the privacy of users in the single-issuer setting, they fall short in settings with multiple issuers; as the signature of a user reveals the public key of her issuer [10, 8, 4]. A workaround is delegatable credentials [3, 7, 13]: these allow a *trusted root authority* to delegate issuing capabilities to a multitude of issuers in such a way that the resulting anonymous signatures only reveal the public key of the root authority and nothing else.

A trusted root authority, however, contradicts the decentralization property inherent to blockchain technology and its applications. Ideally, blockchain issuers are independent participants and users are free to choose any issuer without undermining their privacy.

We present, in this paper, a solution for anonymous credentials that supports multiple issuers without a *root authority*. We use one-out-of-many proofs to prove that a user was issued a credential by one of the *authorized* issuers, and we leverage zero-knowledge proofs to show that the user knows the secret key corresponding to the credential. We define an ideal functionality that captures the requirements of anonymous credentials in the multi-issuer setting, and demonstrate that the proposed solution is secure in the universal composability (UC) framework. We also show that the solution is designed with performance in mind: the size of user signatures is logarithmic in the number of issuers and does not depend on the number of user attributes.

Summary of Contributions. The contributions of this paper are two-fold:

- A formalization of the security requirements of multi-issuer anonymous credentials in the universal composability framework.
- A solution for multi-issuer anonymous credentials without a root authority, which is UC-secure and whose size is *logarithmic* in the number of issuers and *constant* in the number of user attributes.

Outline. The paper is organized as follows: Section 2 introduces ideal functionality $\mathcal{F}$ that formalizes the security requirements of multi-issuer anonymous credentials. Section 3 presents the building blocks whereas Section 4 describes the solution. The security analysis can be found in Section 5. Finally, Section 6 summarizes related work and Section 7 concludes the paper.

Let $\boldsymbol{I} = \{I_1, ..., I_N\}$ denote the set of authorized issuers and $\mathbf{ATT} = \{\mathtt{att}_1, ..., \mathtt{att}_L\}$ denote the set of admissible attributes.
Let CRED and SIG be two empty maps.

- ENROL. On input $(\text{ENROL}, U, \boldsymbol{A})$ from issuer I, with $\boldsymbol{A} = \{a_1, ..., a_L\}$ being values of attributes $\mathbf{ATT} = \{\mathtt{att}_1, ..., \mathtt{att}_L\}$, proceed as follows:
  1. If I is honest and $\text{CRED}[(U, I)] \not\rightarrow \text{NIL}$, then reject. *(An honest issuer will enrol a user only once).*
  2. Output (ENROL, U, I) to $\mathcal{S}$ and wait for response ENROLEND.
  3. Set $\text{CRED}[(U, I)] \leftarrow \boldsymbol{A}$.
  4. Output ENROLEND to I.
- SIGN. On input $(\text{SIGN}, m, \boldsymbol{A}')$ from some user U, proceed as follows:
  1. If all issuers are honest and if $\nexists\ \boldsymbol{A} \leftarrow \text{CRED}[(U, I_i)]$ such that $\boldsymbol{A} \not\supset \boldsymbol{A}'$, then reject.
  2. Send $(\text{SIGN}, m, \boldsymbol{A}')$ to $\mathcal{S}$ and wait for response $(\text{SIGNEND}, \psi)$.
  3. If $\text{SIG}[\psi] \not\rightarrow \text{NIL}$, then reject.
  4. Set $\text{SIG}[\psi] \leftarrow \langle m, \boldsymbol{A}' \rangle$.
  5. Output $(\text{SIGNEND}, \psi)$ to U.
- VERIFY. On input $(\text{VERIFY}, \psi, m, \boldsymbol{A}')$ from party P, proceed as follows:
  1. If $\text{SIG}[\psi] \rightarrow \langle m, \boldsymbol{A}' \rangle$, then output $(\text{VERIFIED}, \mathtt{true})$ to P (*completeness*).
  2. Else if all issuers are honest and all users with attributes $\boldsymbol{A}$ such that $\boldsymbol{A}' \subset \boldsymbol{A}$ are also honest, then output $(\text{VERIFIED}, \mathtt{false})$ to P (*unforgeability*).
  3. Else query $\mathcal{S}$ with message $(\text{VERIFY}, \psi, m, \boldsymbol{A}')$ and output response $(\text{VERIFIED}, \text{BOOL})$.

Fig. 1. Ideal functionality $\mathcal{F}$ for multi-issuer anonymous credentials

2 Security Definition

2.1 Universal Composability

We formalize the security of multi-issuer anonymous credentials in the universal composability (UC) framework [11], which follows the *simulation* paradigm. More specifically, we define an *ideal functionality* $\mathcal{F}$ that captures the security and privacy guarantees of multi-issuer anonymous credentials in the *ideal world*, where honest parties are restricted to communicating their inputs to $\mathcal{F}$ and obtaining from $\mathcal{F}$ the corresponding outputs. The UC framework assumes an environment $\mathcal{Z}$ that provides inputs to protocol participants and receives their outputs. The security of a protocol Π is hence defined with respect to the capability of $\mathcal{Z}$ to distinguish between *ideal-world* executions and *real-world* executions. A real-world execution is an execution of protocol Π in the presence of an adversary $\mathcal{A}$, who controls the network, corrupts parties and freely communicates with environment $\mathcal{Z}$.

More formally, we say that a protocol Π securely realizes ideal functionality $\mathcal{F}$ *iff*, for every adversary $\mathcal{A}$ in the real world, there is a simulator $\mathcal{S}$ in the ideal world such that $\mathcal{Z}$ cannot distinguish the real world (where $\mathcal{A}$ interacts with Π) from the ideal world (where $\mathcal{S}$ interacts with $\mathcal{F}$). In other words, Π offers the same security guarantees as $\mathcal{F}$. Now, given that $\mathcal{F}$ is secure by construction, the security of Π follows.

Security in the UC frameworks guarantees composability; that is, it ensures that security is preserved even when running multiple instances of protocol Π in parallel.

The following section defines ideal functionality $\mathcal{F}$, which reflects the security properties of multi-issuer anonymous credentials in the *static corruption* model.

2.2 Ideal Functionality $\mathcal{F}$

We define ideal functionality $\mathcal{F}$ with respect to a set of authorized issuers $\boldsymbol{I} = \{I_1, ..., I_N\}$ and a set of admissible attributes $\mathbf{ATT} = \{\mathtt{att}_1, ..., \mathtt{att}_L\}$. Issuers produce credentials for users who hold a set of attribute values $\boldsymbol{A} = \{a_1, ..., a_L\}$ each, such that a_i is the value of attribute $\mathtt{att}_i$. A user U with credentials from one of these issuers is able to *anonymously* sign messages; whereas all other participants are able to verify the signatures. $\mathcal{F}$ accommodates these operations by exposing three interfaces ENROL, SIGN and VERIFY, which the participants call to communicate with $\mathcal{F}$ (cf. Figure 1).

Enrol Issuer $I \in \boldsymbol{I}$ enrols user U by calling $\mathcal{F}$ with message $(\text{ENROL}, U, \boldsymbol{A})$. $\mathcal{F}$ in turn notifies simulator $\mathcal{S}$ that an enrollment of user U by I is taking place. If $\mathcal{S}$ agrees to the enrollment, then $\mathcal{F}$ sets $\text{CRED}[(U, I)] \leftarrow \boldsymbol{A}$.

Sign User U signs a message m while disclosing attributes $\boldsymbol{A}' = \{a'_1, ..., a'_n\}$ by sending a request $(\text{SIGN}, m, \boldsymbol{A}')$ to $\mathcal{F}$. If all issuers are honest and U does not have a credential for attribute values $\boldsymbol{A}$ such that $\boldsymbol{A}' \subset \boldsymbol{A}$, then $\mathcal{F}$ rejects the signature request. Otherwise, $\mathcal{F}$ forwards the signature request to $\mathcal{S}$, and in return, receives a string ψ. Next, $\mathcal{F}$ checks if $\text{SIG}[\psi]$ is empty, and if so, sets $\text{SIG}[\psi] \leftarrow \langle m, \boldsymbol{A}' \rangle$.

Verify Finally, to verify a signature ψ on message m relative to attribute values $\boldsymbol{A}'$, a party P calls $\mathcal{F}$ with tuple $(\text{VERIFY}, \psi, m, \boldsymbol{A}')$. If ψ was originally generated for $(m, \boldsymbol{A}')$, then $\mathcal{F}$ returns `true` (this reflects *completeness*). If not, and if all the issuers are honest and all users with attributes encompassing $\boldsymbol{A}'$ are also honest, then $\mathcal{F}$ returns `false` (this captures *unforgeability*). Otherwise, $\mathcal{F}$ sends tuple $(\text{VERIFY}, \psi, m, \boldsymbol{A}')$ to $\mathcal{S}$. $\mathcal{S}$ responds with a Boolean BOOL that $\mathcal{F}$ outputs in turn.

3 Preliminaries

Notation Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of large prime order q, such that an efficient bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ exists. Let $\mathbf{g}$ and $\tilde{\mathbf{g}}$ be random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Let $H : \{0,1\}^* \rightarrow \mathbb{F}_q$ denote a cryptographic hash function.

In the remainder of the paper, lower-case letters in bold font refer to group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$. Group elements with tilde on top refer to elements in $\mathbb{G}_2$ whereas upper-case letters in bold font refer to sets.

Finally, we denote by Λ the tuple $(q, \mathbf{g}, \tilde{\mathbf{g}}, e, H)$. The cryptographic primitives described in this section all take Λ as a default input, which we omit for simplicity purposes.

3.1 Groth Signature

Groth signature [18] is a structure-preserving signature that sign vectors of group elements. It consists of four algorithms SETUP, KEYGEN, SIGN and VERIFY.

- $\text{SETUP}(l) \rightarrow sp$: on input of integer l, SETUP randomly selects l random generators $(\mathbf{h}_1, ..., \mathbf{h}_l)$ of $\mathbb{G}_1^l$ and returns setup parameters $sp = (\mathbf{g}, \tilde{\mathbf{g}}, \mathbf{h}_1, ..., \mathbf{h}_l)$. sp is an implicit input to subsequent algorithms.
- $\text{KEYGEN}(.) \rightarrow (sk, pk)$: KEYGEN randomly selects secret key $sk = x \in \mathbb{F}_q^*$ and computes the corresponding public key $pk = \tilde{\mathbf{x}} = \tilde{\mathbf{g}}^x$.
- $\text{SIGN}(sk, \vec{\mathbf{m}}) \rightarrow \sigma$: on input of secret key $sk = x$ and vector of messages $\vec{\mathbf{m}} = (\mathbf{m}_1, ..., \mathbf{m}_l) \in \mathbb{G}_1^l$, SIGN randomly selects $r \in \mathbb{F}_q^*$ and computes signature $\sigma = (\tilde{\mathbf{r}}, \mathbf{s}, \mathbf{t}_1, ..., \mathbf{t}_l)$ where

$$\tilde{\mathbf{r}} = \tilde{\mathbf{g}}^{1/r} \ ; \ \mathbf{s} = (\mathbf{h}_1 \mathbf{g}^x)^r \ ; \ \mathbf{t}_i = (\mathbf{h}_i^x \mathbf{m}_i)^r, \forall 1 \leq i \leq l$$

- $\text{VERIFY}(pk, \vec{\mathbf{m}}, \sigma) \rightarrow \text{BOOL}$: on input of public key $pk = \tilde{\mathbf{x}}$, messages $\vec{\mathbf{m}} = (\mathbf{m}_1, ..., \mathbf{m}_l) \in \mathbb{G}_1^l$ and signature $\sigma = (\tilde{\mathbf{r}}, \mathbf{s}, \mathbf{t}_1, ..., \mathbf{t}_l)$, VERIFY checks whether the following equations hold:

$$e(\mathbf{s}, \tilde{\mathbf{r}}) = e(\mathbf{h}_1, \tilde{\mathbf{g}}) e(\mathbf{g}, \tilde{\mathbf{x}}) \ ;$$
$$e(\mathbf{t}_i, \tilde{\mathbf{r}}) = e(\mathbf{h}_i, \tilde{\mathbf{x}}) e(\mathbf{m}_i, \tilde{\mathbf{g}}) \ , \forall \ 1 \leq i \leq l \ ;$$

  If so, then VERIFY returns $\text{BOOL} = \mathtt{true}$ signifying that the signature is valid; otherwise, VERIFY returns $\text{BOOL} = \mathtt{false}$.

3.2 Pairing-based Accumulators

Pairing-based accumulators (PACC) operate in pairing groups and comprise the following algorithms.

- CRSGen(L) $\rightarrow$ *acrs*: given the maximum number of values that can be accumulated L, CRSGen picks a random number z in $\mathbb{F}_q^*$ and outputs $acrs = (\mathbf{g}, \mathbf{g}^z, ..., \mathbf{g}^{z^L}, \tilde{\mathbf{g}}, \tilde{\mathbf{g}}^z, ..., \tilde{\mathbf{g}}^{z^L}) = (\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_L, \tilde{\mathbf{z}}_0, \tilde{\mathbf{z}}_1, ..., \tilde{\mathbf{z}}_L) \in \mathbb{G}_1^{L+1} \times \mathbb{G}_2^{L+1}$. All subsequent algorithms take *acrs* as an implicit input.
- Eval($\boldsymbol{A}$) $\rightarrow$ **acc**: on input of a set $\boldsymbol{A} = \{a_1, ..., a_L\}$, Eval computes polynomial $\mathfrak{p}(X) = \prod_{i=1}^{L}(X + a_i) = \sum_{i=0}^{L} \alpha_i X^i$ and outputs $\mathbf{acc} = \prod_{i=0}^{L} \mathbf{z}_i^{\alpha_i}$.
- WitGen($\boldsymbol{A}', \boldsymbol{A}$) $\rightarrow \boldsymbol{\omega}'$: on input of set $\boldsymbol{A}' = \{a'_1, ..., a'_n\}$ and a set $\boldsymbol{A} = \{a_1, ..., a_L\}$, WitGen computes polynomial $\mathfrak{q}(X) = \prod_{i=1 a_i \notin \boldsymbol{A}'}^{L}(X + a_i) = \sum_{i=0}^{L-n} \beta_i X^i$ and outputs witness $\boldsymbol{\omega}' = \prod_{i=0}^{L-n} \mathbf{z}_i^{\beta_i}$.
- Verify($\boldsymbol{A}', \boldsymbol{\omega}', \mathbf{acc}$) $\rightarrow$ bool: on input of set $\boldsymbol{A}' = \{a'_1, ..., a'_n\}$, witness $\boldsymbol{\omega}'$ and accumulator **acc**, Verify computes polynomial $\mathfrak{d}(X) = \prod_{i=1}^{n}(X + a'_i) = \sum_{i=0}^{n} \delta_i X^i$ and checks whether $e(\mathbf{acc}, \tilde{\mathbf{g}}) = e(\boldsymbol{\omega}', \prod_{i=0}^{n} \tilde{\mathbf{z}}_i^{\delta_i})$. If so, then Verify outputs bool = `true` indicating that $\boldsymbol{A}'$ is accumulated in **acc**; else Verify outputs bool = `false`.

Nguyen et al. [22] showed that these accumulators are collision resistant under the L-strong Diffie-Hellman (L-SDH) assumption.

Definition 1 (L-SDH Assumption). *Let z be randomly-chosen element in $\mathbb{F}_q$ and let $\mathbf{g}$ and $\tilde{\mathbf{g}}$ be two random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.*

We say that L-strong Diffie-Hellman assumption holds, if given $(\mathbf{g}, \mathbf{g}^z, ..., \mathbf{g}^{z^L}, \tilde{\mathbf{g}}, \tilde{\mathbf{g}}^z, ..., \tilde{\mathbf{g}}^{z^L})$, it is computationally infeasible to output a pair $(c, \mathbf{g}^{1/z+c}) \in \mathbb{F}_q \times \mathbb{G}_1$.

3.3 Elgamal-based Commitments

For consistency purposes, we describe the algorithms underlying Elgamal-based commitments [14] in group $\mathbb{G}_2$.

- CRSGen(.) $\rightarrow$ *ccrs*: CRSGen randomly selects two generators $ccrs = (\tilde{\mathbf{g}}, \tilde{\mathbf{y}}) \in \mathbb{G}_2$. For simplicity, we omit *ccrs* from subsequent algorithms.
- commit(m) $\rightarrow$ (com, r): on input of message $m \in \mathbb{F}_q$, commit randomly selects $r \in \mathbb{F}_q^*$ and computes $\text{com} = (\tilde{\mathbf{g}}^m \tilde{\mathbf{y}}^r, \tilde{\mathbf{g}}^r)$.
- open(m, r, com) $\rightarrow$ bool: on input of message m, randomness r and Elgamal commitment $\text{com} = (\tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2)$, open checks if $\tilde{\mathbf{c}}_1 = \tilde{\mathbf{g}}^m \tilde{\mathbf{y}}^r$ and $\tilde{\mathbf{c}}_2 = \tilde{\mathbf{g}}^r$. If so, then open returns bool = `true`, indicating that com is a commitment to m. Otherwise, it returns bool = `false`.

These commitments are perfectly binding and computationally hiding under the Decisional Diffie-Hellman assumption.

3.4 One-Out-Of-Many Proof

One-out-of-many proof [19] is a public-coin special-honest verifier zero-knowledge proof of knowledge that enables a prover to show that in a vector $\vec{\text{com}} = (\text{com}_1, ..., \text{com}_N)$ of additively-homomorphic commitments, there is a commitment com_i that opens to 0. The proof is originally interactive, however thanks to the Fiat-Shamir heuristic, it can be made non-interactive.

In this paper, we use the variant introduced by Bootle et al. [6], in which commitments com_i are computed as Elgamal ciphertexts. For ease of exposition, we only provide a high-level description of the algorithms underlying this variant.

One-out-of-many proof (OOMP) involves algorithms:

- CRSGen(.) $\rightarrow$ *ccrs*: CRSGen returns two random generators $ccrs = (\tilde{\mathbf{g}}, \tilde{\mathbf{y}})$. Subsequent algorithms take *ccrs* as an implicit input.
- Prove($\vec{\text{com}}, i, r$) $\rightarrow \phi$: on input of a vector of Elgamal commitments $\vec{\text{com}} = (\text{com}_1, ..., \text{com}_N)$, an index $i \in \{1, N\}$ and randomness $r \in \mathbb{F}_q$, such that $\text{com}_i = (\tilde{\mathbf{g}}^0 \tilde{\mathbf{y}}^r, \tilde{\mathbf{g}}^r)$, Prove outputs a proof ϕ.
- Verify($\vec{\text{com}}, \phi$) $\rightarrow$ bool: on input of a vector of Elgamal commitments $\vec{\text{com}} = (\text{com}_1, ..., \text{com}_N)$ and a proof ϕ, Verify returns a Boolean bool. If bool = `true`, then that indicates that there is a commitment in $\vec{\text{com}}$ that opens to 0.

Let $\mathcal{R}$ be a binary relation whose membership can be efficiently verified.

Let L be an empty map.

- On input (PROVE, pub, w) from some party P such that $\mathcal{R}(pub, w) = 1$, send (PROVE, pub) to $\mathcal{S}$.
- On input $(\text{PROVEEND}, \xi)$ from $\mathcal{S}$, set $L[\langle pub, \xi \rangle] \leftarrow \texttt{true}$ and send $(\text{PROVEEND}, \xi)$ to P.
- On input $(\text{VERIFY}, pub, \xi)$ from some party P, proceeds as follows:
  1. If $L[\langle pub, \xi \rangle] \rightarrow \text{NIL}$, then output $(\text{VERIFY}, pub, \xi)$ to $\mathcal{S}$ and wait for response $(\text{WITNESS}, w)$. If $\mathcal{R}(pub, w) = 1$, then set $L[\langle pub, \xi \rangle] \leftarrow \texttt{true}$; else set $L[\langle pub, \xi \rangle] \leftarrow \texttt{false}$.
  2. Return $L[\langle pub, \xi \rangle]$.

Fig. 2. Non-interactive zero-knowledge functionality $\mathcal{F}_{\text{NIZK}}$ parametrized with relation $\mathcal{R}$. Based on the one described by Groth et al. [20].

3.5 Schnorr Signatures

Schnorr signatures [25] consist of three algorithms KEYGEN, SIGN and VERIFY.

- $\text{KEYGEN}(.) \rightarrow (sk, pk)$: upon call, KEYGEN first picks a generator $\mathbf{u} \in \mathbb{G}_1$, then randomly selects a secret key v and computes $\mathbf{v} = \mathbf{u}^v$. KEYGEN concludes by outputting $sk = v$ and $pk = (\mathbf{u}, \mathbf{v})$.
- $\text{SIGN}(sk, pk, m) \rightarrow \gamma$: on input of secret key $sk = v \in \mathbb{F}_q$, public key $pk = (\mathbf{u}, \mathbf{v})$ and message $m \in \{0,1\}^*$, SIGN randomly selects $f \in \mathbb{F}_q$, computes $\mathbf{f} = \mathbf{u}^f$, $c = H(m, pk, \mathbf{f})$ and $\pi = f + cv \mod q$, and returns $\gamma = (\mathbf{f}, \pi)$.
- $\text{VERIFY}(pk, m, \gamma) \rightarrow \text{BOOL}$: on input of message m, signature $\gamma = (\mathbf{f}, \pi)$ and public key $pk = (\mathbf{u}, \mathbf{v})$, VERIFY computes $c = H(m, pk, \mathbf{f})$ and checks if $\mathbf{v}^c\mathbf{f} = \mathbf{u}^\pi$. If so, VERIFY returns $\text{BOOL} = \texttt{true}$, meaning that γ is a valid signature under public key pk. Else VERIFY returns $\text{BOOL} = \texttt{false}$.

3.6 Non-interactive Zero-Knowledge Proofs (NIZK)

For clarity purposes, we introduce notations and definitions related to non-interactive zero-knowledge proofs (NIZK for short) that we use in our solution.

Let $\mathcal{R}$ be a binary relation. For pairs (w, pub) with $\mathcal{R}(w, pub) = 1$, we call w witness (i.e. private input) and pub statement (i.e. public input).

Accordingly, a NIZK for $\mathcal{R}$ is defined by the following algorithms:

- $\text{CRSGEN}(\mathcal{R}) \rightarrow crs$: on input of binary relation $\mathcal{R}$, CRSGEN outputs a common reference string crs. For simplicity, we omit crs from the inputs of PROVE and VERIFY.
- $\text{PROVE}(w, pub) \rightarrow \xi$: on input of pair (w, pub) such that $\mathcal{R}(w, pub) = 1$, PROVE outputs a proof ξ.
- $\text{VERIFY}(pub, \xi) \rightarrow \text{BOOL}$: on input of public input pub and proof ξ, VERIFY outputs a Boolean BOOL. $\text{BOOL} = \texttt{true}$ implies that ξ is a valid proof relative to public input pub and relation $\mathcal{R}$.

A NIZK is said to be *correct* if honestly-generated proofs are always accepted by VERIFY. It is said to be *knowledge extractable* if it is infeasible for an adversary with access to an arbitrary number of proofs to produce a valid proof without access to a valid witness. Finally, we say that NIZK is zero-knowledge if the verification of valid proofs yields nothing beyond their validity.

For ease of exposition, we introduce an ideal functionality $\mathcal{F}_{\text{NIZK}}$ (cf. Figure 2) as a stand in for knowledge extractable NIZK. Details on how to instantiate $\mathcal{F}_{\text{NIZK}}$ are deferred to Appendix B.

4 Solution

Overview To sign a message m anonymously while disclosing some attributes $\boldsymbol{A}' = \{a'_1, ..., a'_n\}$, a user first uses *one-out-of-many proof* to show that an *Elgamal-encrypted* public key is the public key of one of the authorized issuers. Afterwards, the user proves in zero-knowledge that she knows a valid *Groth signature* under the encrypted public key on a vector composed of a Schnorr public key and a set of attributes $\boldsymbol{A} = \{a_1, ..., a_L\}$ such that $\boldsymbol{A}' \subset \boldsymbol{A}$. Finally, the user leverages *Schnorr signatures* to sign message m while proving knowledge of the secret key underlying the signed Schnorr public key.

To make sure that the size of users' signatures do not grow in the number of attributes, we use – similar to [24], *pairing-based accumulators*. Note that accumulators enable constant-size proofs of membership that translates to signatures whose size is constant in the number of attributes.

Let

$$sp = (\mathbf{g}, \tilde{\mathbf{g}}, \mathbf{h}_1, \mathbf{h}_2) \leftarrow \text{GROTH.SETUP}(2)\ ,\ ccrs \leftarrow \text{ELGAMAL.CRSGEN}(.)\ ,\ \ acrs \leftarrow \text{PACC.CRSGEN}(L)$$

Let

$$w = (upk_2, ipk_i, r, t', \mathbf{s}, \mathbf{t}_1, \mathbf{t}_2, \boldsymbol{\omega}', \mathbf{acc}) \quad (1)$$

$$pub = (upk', \boldsymbol{A}', \text{COM}_i, \tilde{\mathbf{r}}, \mathbf{h}_1, \mathbf{h}_2, \tilde{\mathbf{g}}, ccrs, acrs) \text{ such that: } \ upk' = (upk'_1, upk'_2) \quad (2)$$

Finally, let $\mathcal{R}$ be a binary relation such that $\mathcal{R}(w, pub) = 1$ iff

$$\text{COM}_i \leftarrow \text{ELGAMAL.ENC}(ipk_i, r, ccrs);\ (upk'_1, upk'_2) = (\mathbf{g}^{t'}, upk_2^{t'})$$

$$e(\mathbf{s}, \tilde{\mathbf{r}}) = e(\mathbf{h}_1, \tilde{\mathbf{g}})e(\mathbf{g}, ipk_i);\ e(\mathbf{t}_1, \tilde{\mathbf{r}}) = e(\mathbf{h}_1, ipk_i)e(upk_2, \tilde{\mathbf{g}});\ e(\mathbf{t}_2, \tilde{\mathbf{r}}) = e(\mathbf{h}_2, ipk_i)e(\mathbf{acc}, \tilde{\mathbf{g}});$$

$$\forall 1 \leq i \leq m : \text{PACC.VERIFY}(\boldsymbol{A}', \boldsymbol{\omega}', \mathbf{acc}, acrs) \rightarrow \texttt{true};$$

Fig. 3. Binary relation $\mathcal{R}$

4.1 Description

Our protocol Π for multi-issuer anonymous credentials comprises four phases:

Setup Authorized issuers agree on the set of admissible attributes $\mathbf{ATT} = \{\texttt{att}_1, ..., \texttt{att}_L\}$ and on the public parameters (see Figure 6).

Afterwards, each authorized issuer I_i runs GROTH.KEYGEN(.) to produce a Groth signature key pair (isk_i, ipk_i) and calls registration functionality $\mathcal{F}_{\text{REG}}$ to register. $\mathcal{F}_{\text{REG}}$ accepts the registration request only if the provided key pair is valid and was not registered before (check Figure 4).

SETUP concludes when all authorized issuers are registered.

Let VALIDATE be an algorithm that takes as input a key pair (sk, pk) and outputs `true` if the key pair is a valid Groth signature key pair and `false` otherwise.

Let REGISTERED and PK be two empty maps.

- On input (REGISTER, (sk, pk)) from party P, send (REGISTER, P, pk) to $\mathcal{S}$.
- On input of response (OK, P, pk) from $\mathcal{S}$, do:
  1. If REGISTERED$[pk] \rightarrow$ `true`, then reject.
  2. If VALIDATE$(sk, pk) \rightarrow$ `false`, then reject.
  3. Set PK$[P] \leftarrow pk$ and REGISTERED$[pk] \leftarrow$ `true`.
  4. Output (REGISTEREND, pk) to P.
- On input (LOOKUP, P) from some party P', return (LOOKUPEND, P, PK$[P]$).

Fig. 4. Registration functionality $\mathcal{F}_{\text{REG}}$ parametrized with Groth signature.

Enrol At the beginning of this phase, user U produces a key pair (usk, upk) where $upk = (upk_1, upk_2) = (\mathbf{g}, \mathbf{g}^{usk})$.

U then contacts authorized issuer I_i to obtain a credential that binds public key upk to U's attributes $\boldsymbol{A} = \{a_1, ..., a_L\}$. Accordingly, I_i first checks if U is already enrolled and if public key upk is already registered. If not, I_i verifies if U actually possesses attributes $\boldsymbol{A} = \{a_1, ..., a_L\}$. If this verification succeeds, then I_i engages with U in an interactive protocol that allows U to prove knowledge of secret key usk (e.g., U signs a nonce from I_i using usk and Schnorr signature). If the interactive protocol terminates successfully, then I_i generates U's credential, which corresponds to CRED $= (usk, upk_1, upk_2, \boldsymbol{A}, \sigma)$ with $\sigma \leftarrow \text{SIGN}(isk_i, \vec{\mathbf{m}})$, $\vec{\mathbf{m}} = (upk_2, \mathbf{acc})$ and $\mathbf{acc} \leftarrow \text{PACC.EVAL}(\boldsymbol{A})$.

During this phase, U and I_i use functionality $\mathcal{F}_{\text{SMT}}$ (see Figure 5) to exchange messages securely and privately (i.e., no one can tamper with or access the content of the exchanged messages).

Let S and R denote sender and recipient respectively.
Let m denote the message to be exchanged between S and R.

- On input (SEND, R, m) from S:
  - If both S and R are honest, then send (SEND, S, R) to $\mathcal{S}$.
  - Else send (SEND, S, R, m) to $\mathcal{S}$.
- On input $(\text{RECEIVE}, S, R)$ from $\mathcal{S}$, output $(\text{RECEIVE}, S, m)$ to R.

Fig. 5. Secure message transmission functionality $\mathcal{F}_{\text{SMT}}$. Based on the functionality in [12].

Sign User U is endowed with a credential $\text{CRED} = (usk, upk, \boldsymbol{A}, \sigma)$[1] s.t. $upk = (upk_1, upk_2) = (\mathbf{g}, \mathbf{g}^{usk})$, $\text{VERIFY}(ipk_i, \vec{\mathbf{m}}, \sigma) \rightarrow \texttt{true}$ for $\vec{\mathbf{m}} = (upk_2, \mathbf{acc})$ and $\mathbf{acc} \leftarrow \text{PACC.EVAL}(\boldsymbol{A})$.

To sign a message m while disclosing attributes $\boldsymbol{A'}$, U first calls $\mathcal{F}_{\text{REG}}$ to retrieve public keys $\{ipk_1, ..., ipk_N\}$ of authorized issuers $\boldsymbol{I} = \{I_1, ..., I_N\}$. Then U computes an Elgamal ciphertext COM_i of public key ipk_i and produces a one-out-of-many-proof ϕ that demonstrates that COM_i actually encrypts the public key of one of the authorized issuers. Next, U randomizes public key upk to upk' and computes witness $\boldsymbol{\omega'} \leftarrow \text{PACC.WITGEN}(\boldsymbol{A'}, \boldsymbol{A})$. U then calls $\mathcal{F}_{\text{NIZK}}$ to generate a zero-knowledge proof ξ that proves that U knows a tuple $(upk_2, \mathbf{acc}, \sigma)$ such that:

1. σ is a valid Groth signature of $\vec{\mathbf{m}} = (upk_2, \mathbf{acc})$ under the public key encrypted in COM_i;
2. $\boldsymbol{A'}$ is accumulated in $\mathbf{acc}$;
3. upk' is a randomization of $(\mathbf{g}, upk_2)$.

Finally, U computes a Schnorr signature

$$\gamma' \leftarrow \text{SCHNORR.SIGN}(usk, upk', m||\xi)$$

We denote ψ the output of this phase. ψ contains encryption COM_i, Schnorr signature γ', randomized public key upk', one-out-of-many-proof ϕ and zero-knowledge proof ξ.

Verify Given a message m, disclosed attributes $\boldsymbol{A'}$ and a signature ψ, a party P retrieves from ψ signature γ', randomized public key upk', ciphertext COM_i, one-out-of-many-proof ϕ and zero-knowledge proof ξ.

P verifies if γ' is a valid Schnorr signature of message $m||\xi$ relative to upk'. If not, then P rejects. Next, P queries functionality $\mathcal{F}_{\text{REG}}$ to retrieve the public keys of the authorized issuers. Thanks to ϕ, P verifies whether COM_i is a valid Elgamal encryption of one of the retrieved public keys. If not, then P rejects. Finally, P calls $\mathcal{F}_{\text{NIZK}}$ to verify if ξ is a valid zero-knowledge proof of the statement described in Figure 3.

A more formal description of the protocol can be found in Figure 6.

4.2 Performance

Relying on one-out-of-many proofs to show that a user obtained a credential from one of the authorized issuers results in a proof whose size is logarithmic in the number of issuers. More precisely, one-out-of-many proof yields $4\log(N)$ elements in $\mathbb{G}_2$ and $1 + 3\log(N)$ elements in $\mathbb{F}_q$), with N being the number of issuers. On the other hand, The computation and verification of the proof consists of $(7 + 2N)\log(N) + 2$ and $2N + 12\log(N) + 2$ exponentiations in $\mathbb{G}_2$ respectively. Given that the number of issuers is generally in the order of tens, the $O(N\log(N))$ complexity will be acceptable for most applications.

Appendix B details how to instantiate $\mathcal{F}_{\text{NIZK}}$ to guarantee security in the UC framework. The instantiation relies on Groth-Sahai proofs [21, 15], and accordingly, a user proves that they hold certain attributes by performing 74 and 32 exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. The resulting proof consists of 43 and 21 elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ and 4 elements in $\mathbb{F}_q$ and its Verification corresponds to computing $((11 + 2N)\log(N) + 2)$ exponentiations in $\mathbb{G}_2$ and 69 pairings. Performance however can be further optimized using batching techniques that reduce the number of pairings in the verification, see [5]. Table 1 summarizes our performance numbers.

Note that if we forgo UC security, then we can use Schnorr proofs instead of Groth-Sahai proofs; this yields much more efficient proofs (in terms of both proof generation and verification). In [7], Camenisch et al. show how to extend Schnorr proofs to prove statements about elements in bilinear groups.

[1] σ is randomized for every signature computation.

Let $\mathbf{ATT} = \{\mathtt{att}_1, ..., \mathtt{att}_L\}$ denote the set of admissible attributes and $\boldsymbol{I} = \{I_1, ..., I_N\}$ denote the set of authorized issuers.

0. SETUP. In the setup phase, issuers in $\boldsymbol{I}$ agree on the system public parameters. These are defined as

$$pp = (\Lambda, sp, ccrs, acrs)$$

whereby $\Lambda = (q, \mathbf{g}, \tilde{\mathbf{g}}, e, H)$, $sp \leftarrow$ GROTH.SETUP(2), $ccrs \leftarrow$ ELGAMAL.CRSGEN(.) and $acrs \leftarrow$ PACC.CRSGEN(L).
Issuers also parametrize ideal functionality $\mathcal{F}_{\text{NIZK}}$ with relation $\mathcal{R}$ defined in Figure 3.
Finally, each issuer $I_i \in \boldsymbol{I}$ calls $\mathcal{F}_{\text{REG}}$ to register its Groth signature key pair $(isk_i, ipk_i) \leftarrow$ GROTH.KEYGEN(.).

1. ENROL. User U generates a key pair (usk, upk) with $upk = (upk_1, upk_2) = (\mathbf{g}, \mathbf{g}^{usk})$. U then requests, from issuer $I_i \in \boldsymbol{I}$, a credential that binds upk to her attributes $\boldsymbol{A} = \{a_1, ..., a_L\} \in \mathbb{F}_q^L$.
I_i checks if upk is already registered. If not, then it checks if U actually holds attributes $\boldsymbol{A} = \{a_1, ..., a_L\}$. If so, it sends a nonce $\mathfrak{n}$ to U, who responds with $\gamma \leftarrow$ SCHNORR.SIGN($usk, upk, \mathfrak{n}$). Upon receipt of γ, I_i checks if SCHNORR.VERIFY($upk, \mathfrak{n}, \gamma$) $\rightarrow$ `true`. If not, then I_i rejects; else it executes the following:
   - run $\mathbf{acc} \leftarrow$ PACC.EVAL($\boldsymbol{A}, acrs$);
   - compute $\sigma \leftarrow$ GROTH.SIGN($isk_i, \vec{\mathbf{m}}$) for $\vec{\mathbf{m}} = (upk_2, \mathbf{acc})$;
   - output σ to U.

   On receiving σ, U computes $\mathbf{acc} \leftarrow$ PACC.EVAL($\boldsymbol{A}, acrs$) and executes BOOL $\leftarrow$ GROTH.VERIFY($ipk_i, \vec{\mathbf{m}}, \sigma$) for $\vec{\mathbf{m}} = (upk_2, \mathbf{acc})$. If BOOL = `false`, then U rejects the signature; otherwise, she stores credential CRED = $(usk, upk, \boldsymbol{A}, \sigma)$.
   User U and issuer I_i leverage functionality $\mathcal{F}_{\text{SMT}}$ to exchange messages.

2. SIGN. We assume that user U has a credential CRED = $(usk, upk, \boldsymbol{A}, \sigma)$ such that σ is a valid Groth signature under public key ipk_i, and that every time U wishes to sign a message she randomizes σ.
   U signs a message $m \in \{0,1\}^*$ and proves that she holds attributes $\boldsymbol{A}' \subset \boldsymbol{A}$ as follows.
   - using $\mathcal{F}_{\text{REG}}$, lookup the public keys $(ipk_1, ..., ipk_N)$ of the authorized issuers;
   - parse $ccrs$ as $(\tilde{\mathbf{g}}, \tilde{\mathbf{y}})$;
   - pick $r \in \mathbb{F}_q^*$ and compute $\text{COM}_i = (\tilde{\mathbf{c}}_{i,1}, \tilde{\mathbf{c}}_{i,2}) = (ipk_i \tilde{\mathbf{y}}^r, \tilde{\mathbf{g}}^r)$;
   - for all $1 \leq j \leq N$, compute $\text{COM}'_j = (\tilde{\mathbf{c}}_{j,1}, \tilde{\mathbf{c}}_{j,2}) = (\tilde{\mathbf{c}}_{i,1}/ipk_j, \tilde{\mathbf{c}}_{i,2})$;
   - run $\phi \leftarrow$ OOMP.PROVE($\vec{\text{COM}}', i, r, ccrs$) for $\vec{\text{COM}}' = (\text{COM}'_1, ..., \text{COM}'_N)$;
   - pick $t' \in \mathbb{F}_q^*$ and compute $upk' = (upk'_1, upk'_2) = (\mathbf{g}^{t'}, upk_2^{t'})$;
   - parse Groth setup parameters sp from public parameters pp as $(\mathbf{g}, \tilde{\mathbf{g}}, \mathbf{h}_1, \mathbf{h}_2)$;
   - parse σ from CRED as $(\tilde{\mathbf{r}}, \mathbf{s}, \mathbf{t}_1, \mathbf{t}_2) \in \mathbb{G}_2 \times \mathbb{G}_1^3$;
   - compute $\boldsymbol{\omega}' \leftarrow$ PACC.WITGEN($\boldsymbol{A}', \boldsymbol{A}$);
   - call $\mathcal{F}_{\text{NIZK}}$ with (PROVE, pub, w) where w and pub are defined in Figure 3, Equations (1) and (2), and satisfy relation $\mathcal{R}$ (this results in proof ξ);
   - compute $\gamma' \leftarrow$ SCHNORR.SIGN($usk, upk', m||\xi$);
   - finally, output message m, attributes $\boldsymbol{A}'$ and the corresponding signature $\psi = (upk', \gamma', \text{COM}_i, \phi, \tilde{\mathbf{r}}, \xi)$

3. VERIFY. In the verification phase, a verifier who is given message m, attributes $\boldsymbol{A}'$ and signature ψ executes the following steps:
   - parse ψ as tuple $(upk', \gamma', \text{COM}_i, \phi, \tilde{\mathbf{r}}, \xi)$;
   - run BOOL $\leftarrow$ SCHNORR.VERIFY($upk', m||\xi, \gamma'$); if BOOL = `false`, then reject;
   - parse COM_i as pair $(\tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2)$;
   - using $\mathcal{F}_{\text{REG}}$, lookup the public keys $(ipk_1, ..., ipk_N)$ of the authorized issuers;
   - for all $1 \leq j \leq N$, compute $\text{COM}'_j = (\tilde{\mathbf{c}}_{j,1}, \tilde{\mathbf{c}}_{j,2}) = (\tilde{\mathbf{c}}_1/ipk_j, \tilde{\mathbf{c}}_2)$;
   - run BOOL $\leftarrow$ OOMP.VERIFY($\vec{\text{COM}}', \phi, ccrs$) for $\vec{\text{COM}}' = (\text{COM}'_1, ..., \text{COM}'_N)$; if BOOL = `false`, then reject;
   - else, call $\mathcal{F}_{\text{NIZK}}$ with (pub, ξ) (where pub is defined in Figure 3 – Equation (2)) and output the response.

Fig. 6. Protocol Π for multi-issuer anonymous credentials.

Signature Generation	Signature Verification	Signature Size	Assumptions
$68\,\text{exp}$ in $\mathbb{G}_1$ + $(7+2N)\log(N)+40\,\text{exp}$ in $\mathbb{G}_2$	$2N+12\log(N)+2\,\text{exp}$ in $\mathbb{G}_2$ 69 pairings	37 $\mathbb{G}_1$ + $(27+4\log(N))$ $\mathbb{G}_2$ + $(5+3\log(N))\mathbb{F}_q$	DDH + L-SDH + Random Oracle

Table 1. Performance numbers of anonymous credentials without a root authority. We do not account for the cost incurred by the Schnorr signature. In terms of size, Schnorr signature adds 1 element in $\mathbb{F}_q$ and 3 elements in $\mathbb{G}_1$. In terms of proof generation and verification, it adds 3, respectively 2, exponentiations in $\mathbb{G}_1$.

In the same vein, the one-out-of-many proof we use can also be replaced by any other primitive that admits Elgamal-based commitments as input.

5 Security Analysis

In this section we prove the main theorem of the paper.

Theorem 1. *Protocol Π realizes ideal functionality $\mathcal{F}$.*

We use a series of *indistinguishable* games (**Game 0** to **Game 6**) to prove that environment $\mathcal{Z}$ cannot tell apart the real world (where it interacts with protocol Π) and the ideal world (where it interacts with functionality $\mathcal{F}$ and simulator $\mathcal{S}$).

Game 0 corresponds to an execution of protocol Π in the real world whereas **Game 6** describes the interactions of the participants with ideal functionality $\mathcal{F}$. **Game 1** and **Game 2** allow $\mathcal{S}$ to run Π on behalf of the honest parties, while **Game 3** introduces a dummy functionality $\mathcal{F}'$ that we gradually update in subsequent games so that we end up with functionality $\mathcal{F}$ in **Game 6**.

Game 0. This corresponds to an execution of protocol Π.

Game 1. Simulator $\mathcal{S}$ simulates hybrid functionalities $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SMT}}$ and $\mathcal{F}_{\text{NIZK}}$ correctly. This is equal to **Game 0**.

Game 2. Simulator $\mathcal{S}$ runs protocol Π on behalf of all honest parties. That is, $\mathcal{S}$ receives the inputs of honest parties, and correctly, generates the corresponding outputs. This is indistinguishable from **Game 1**.

Game 3. Simulator $\mathcal{S}$ works together with a functionality $\mathcal{F}'$ to execute protocol Π on behalf of honest parties. $\mathcal{F}'$ is a dummy functionality that forwards inputs from the honest parties to $\mathcal{S}$ and $\mathcal{S}$'s outputs to the honest parties. $\mathcal{S}$ acts as before except that it sends its outputs to $\mathcal{F}'$ instead of sending them directly to the honest parties.

Game 4. Functionality $\mathcal{F}'$ handles ENROL queries from honest issuers the same way as functionality $\mathcal{F}$. If the user is honest, $\mathcal{S}$ does not learn the attribute values of the user. As a result, $\mathcal{S}$ uses dummy values to simulate Π. Since the user and the issuer use $\mathcal{F}_{\text{SMT}}$ to communicate, this change is not detected by environment $\mathcal{Z}$. If the user is corrupt, then $\mathcal{S}$ learns the values of the attributes and can use these to run protocol Π.

Note that $\mathcal{S}$ is able to emulate honest issuers for protocol Π thanks to $\mathcal{F}_{\text{REG}}$. The latter enables $\mathcal{S}$ to extract the secret keys of the issuers at time of registration.

Game 5. Functionality $\mathcal{F}'$ takes care of SIGN requests $(\text{SIGN}, m, \boldsymbol{A}')$ from honest users. If all issuers are honest, then $\mathcal{F}'$ checks if the user is allowed to sign the message relative to the disclosed attributes $\boldsymbol{A}'$. If not, then $\mathcal{F}'$ aborts.

If the user is allowed to sign the message or one of the issuers is corrupt, then $\mathcal{F}'$ forwards request $(\text{SIGN}, m, \boldsymbol{A})$ to $\mathcal{S}$.

Upon receipt of the signature request, $\mathcal{S}$ emulates one of the authorized issuers and generates a credential CRED for attributes $\boldsymbol{A}$ such that $\boldsymbol{A} \supset \boldsymbol{A}'$.

Now using CRED, $\mathcal{S}$ produces a valid signature ψ on message m relative to the disclosed attributes $\boldsymbol{A}'$. $\mathcal{S}$ outputs ψ to $\mathcal{F}'$, which in turn records it and outputs it to the user.

This game is indistinguishable from a **Game 4** except for the following:

1. $\mathcal{F}'$ aborts when all issuers are honest and the user was never issued a credential for attributes $\boldsymbol{A}'$. Since the user is honest, this never happens: an honest user will not request a signature for a message m under attributes $\boldsymbol{A}'$ unless she received a valid credential for those attributes.
2. $\mathcal{S}$ generates a new credential for the user every time it receives a signature request – leveraging an arbitrary issuer. In **Game 4** though, this is not the case: $\mathcal{S}$ reuses the credentials of the user to compute the signatures. Thanks to the semantic security of Elgamal, the encryption of the public key of the issuer does not reveal which issuer was used to produce the credentials. Moreover, under decisional Diffie-Hellman assumption, the randomized public key in the signature does not leak any information about the original public key of the user. Moreover, the witness indistinguishability of one-out-of-many proofs, Schnorr signatures and $\mathcal{F}_{\text{NIZK}}$ guarantees that environment $\mathcal{Z}$ cannot tell which credential was used to sign the message.
   In particular, $\mathcal{Z}$ cannot distinguish between the case where the credential is reused and the case where it is produced afresh.

From (1) and (2) we conclude that **Game 4** and **Game 5** are indistinguishable.

Game 6. In this game, functionality $\mathcal{F}'$ handles VERIFY requests $(\text{VERIFY}, m, \boldsymbol{A}', \psi)$ as follows:

1. If one of the issuers is corrupt, $\mathcal{F}'$ forwards $(\text{VERIFY}, m, \boldsymbol{A}', \psi)$ to $\mathcal{S}$, which then runs Π to verify if signature ψ is a valid signature on message m relative to disclosed attribute values $\boldsymbol{A}'$. $\mathcal{S}$ returns the result of the verification to $\mathcal{F}'$, which in turn outputs it.
2. If there is a corrupt user with attribute values $\boldsymbol{A}$ such that $\boldsymbol{A} \supset \boldsymbol{A}'$, then $\mathcal{F}'$ proceeds like (1).
3. Otherwise, $\mathcal{F}'$ checks if $(m, \boldsymbol{A}', \psi)$ is recorded locally. If so, then $\mathcal{F}'$ outputs `true`; else it outputs `false`.

Note that in (1) and (2), the output of $\mathcal{F}'$ is indistinguishable from **Game 5**. Note also that when $\mathcal{F}'$ returns `true` in (3), **Game 5** will correspondingly outputs `true`. Now what remains to prove is that when $\mathcal{F}'$ outputs `false`, **Game 5** also outputs `false`.

Assume the contrary, that is, there is an adversary $\mathcal{A}$ who is able to produce a signature $(m, \boldsymbol{A}', \psi)$ that passes the verification following Π but not when calling $\mathcal{F}'$.

We recall that in this scenario all issuers are honest and that all users with attributes $\boldsymbol{A}$ such that $\boldsymbol{A} \supset \boldsymbol{A}'$ are also honest.

We show in what follows that under the soundness of one-out-of-many proofs and the collision-resistance of polynomial-based accumulators, this is tantamount to adversary $\mathcal{A}$ breaking the unforgeability of either Schnorr signatures or Groth signatures.

In fact, the soundness of one-out-of-many proof ensures that the commitment COM_i used in relation $\mathcal{R}$ is an Elgamal encryption of the public key of one of the honest issuers (see Figure 3). Using $\mathcal{F}_{\text{NIZK}}$, $\mathcal{S}$ extracts witness

$$w = (usk, upk_2, ipk_i, r, t', \mathbf{s}, \mathbf{t}_1, \mathbf{t}_2, \mathbf{acc}, \boldsymbol{\omega}')$$

for relation $\mathcal{R}$ and public input

$$pub = (upk', \boldsymbol{A}', \text{COM}_i, \tilde{\mathbf{r}}, \mathbf{h}_1, \mathbf{h}_2, \tilde{\mathbf{g}}, ccrs, acrs)$$

Let I_i denote the issuer with public key ipk_i.

- I_i already issued a credential for some user U with public key upk and attributes $\boldsymbol{A}$ such that **acc** is the accumulator of $\boldsymbol{A}$.
  1. $\boldsymbol{A}' \not\subset \boldsymbol{A}$: If U generates a valid signature for message m under attributes $\boldsymbol{A}'$, then U is able to find two sets $\boldsymbol{A} \neq \boldsymbol{A}^*$ that accumulate to the same value. (Notice that $\boldsymbol{A}' \not\subset \boldsymbol{A}$ and $\boldsymbol{A}' \subset \boldsymbol{A}^*$). This breaks the collision resistance of pairing-based accumulators.
  2. $\boldsymbol{A}' \subset \boldsymbol{A}$: This entails that the credential was issued for an honest user U. If the signature was produced by U, then $\mathcal{F}$ will recognize it and output `true`. If the signature was produced by a corrupt user U^*, then U^* is able to impersonate U and generates a valid Schnorr signature γ' for message $m' = m||\xi$, under the randomized public key upk' of honest user U. This breaks the strong unforgeability of Schnorr signatures (cf. Lemma 1).
- I_i did not issue a credential for attributes $\boldsymbol{A}$ that accumulate to **acc**. This implies that tuple $\sigma = (\tilde{\mathbf{r}}, \mathbf{s}, \mathbf{t}_1, \mathbf{t}_2)$ is a valid Groth signature under public key ipk_i that was not generated by I_i (i.e. σ is a forgery).

This implies that **Game 6** and **Game 5** are indistinguishable.

Finally, since $\mathcal{F}' = \mathcal{F}$, this proves that protocol Π realizes $\mathcal{F}$.

Lemma 1. *Let $pk = (\mathbf{u}, \mathbf{v}) \in \mathbb{G}_1 \times \mathbb{G}_1$ be a Schnorr signature public key.*

If there is an adversary $\mathcal{A}'$ who produces a tuple (pk', ξ', m', γ') such that

1. *$pk' = (\mathbf{u}', \mathbf{v}') \in \mathbb{G}_1 \times \mathbb{G}_1$;*
2. *ξ' is a knowledge-extractable zero knowledge proof for binary relation $\mathcal{R}(pub, w)$ such that $pub = (pk, pk')$, $w = t' \in \mathbb{F}_q$, and $\mathcal{R}(pub, w) = 1$ iff $pk' = pk^t$ (i.e., $(\mathbf{u}', \mathbf{v}') = (\mathbf{u}^t, \mathbf{v}^t)$);*
3. *γ' is a valid Schnorr signature of message $m'||\xi'$ under public key pk'.*

then there exists another adversary $\mathcal{A}$ that runs $\mathcal{A}'$ as a subroutine and breaks the strong unforgeability of Schnorr signature under public key pk.

Proof of Lemma 1 is deferred to Appendix A.

6 Related Work

Anonymous credentials allow users to prove statements about themselves without disclosing their identities. Early work [10, 4, 8] hides the identity of the user, however, it reveals the identity of the issuer, making them only suitable for single-issuer settings. A workaround is delegatable credentials[3, 7, 13]: these allow a root authority to delegate issuing capabilities to intermediate issuers in such a way that proving possession of a valid credential only reveals the identity of the root authority and nothing else. Yet, in decentralized settings such as blockchain, it is desirable to enable anonymity without relying on a root authority.

Garman et al. [16] devised a scheme for anonymous credentials that leverages the blockchain to achieve decentralization. Users submit commitments to their secret keys and attributes to the blockchain and sign messages by showing that their credential is stored in the blockchain without revealing which one. More precisely, users produce privacy-preserving proofs of membership using RSA accumulators. The caveat though is that users are required to keep an up-to-date version of their witness, leading to a work linear in the number of user commitments albeit spread-out.

Coconut [26] proposes a solution for federated credential issuance. It relies on a blind variant of threshold Pointcheval-Sanders signatures [23] to allow a group of issuers to collectively generate signatures for users in such a way that users can sign messages anonymously. The setting targeted by Coconut assumes interaction between issuers to compute the secret shares, whereas our solution addresses situations where issuers are independent of each other and never interact.

7 Conclusion

This paper introduces a new solution for multi-issuer anonymous credentials without a root authority. The proposed solution relies on one-out-of-many proof to allow a user to prove that they received a credential from one of the authorized issuers without revealing which one exactly. Additionally, it optimizes the size of user signatures by leveraging Groth signatures and pairing-based accumulators. Finally, the solution is shown to be secure in the universal composability framework.

References

1. https://www.hyperledger.org/use/fabric.
2. https://hyperledger-fabric.readthedocs.io/en/release-2.2/idemix.html.
3. Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 108–125, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
4. Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *Theory of Cryptography*, pages 356–374, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
5. Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch groth–sahai. In Jianying Zhou and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 218–235, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
6. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. Short accountable ring signatures based on ddh. In Günther Pernul, Peter Y A Ryan, and Edgar Weippl, editors, *Computer Security – ESORICS 2015*, pages 243–265, Cham, 2015. Springer International Publishing.
7. Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical uc-secure delegatable credentials with attributes and their application to blockchain. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 683–699, New York, NY, USA, 2017. Association for Computing Machinery.
8. Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. *ACM Trans. Inf. Syst. Secur.*, 15(1), March 2012.
9. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. Cryptology ePrint Archive, Report 2008/539, 2008. https://ia.cr/2008/539.
10. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 56–72, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
11. Ran Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.
12. Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. Cryptology ePrint Archive, Report 2002/059, 2002. https://ia.cr/2002/059.
13. Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 535–555, Cham, 2019. Springer International Publishing.
14. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.
15. Alex Escala and Jens Groth. Fine-tuning groth-sahai proofs. In Hugo Krawczyk, editor, *Public-Key Cryptography – PKC 2014*, pages 630–649, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
16. Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. Cryptology ePrint Archive, Report 2013/622, 2013. https://ia.cr/2013/622.
17. Essam Ghadafi, Nigel. P. Smart, and Bogdan Warinschi. Groth–sahai proofs revisited. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography – PKC 2010*, pages 177–192, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
18. Jens Groth. Efficient fully structure-preserving signatures for large messages. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, pages 239–259, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
19. Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 253–280, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
20. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3), jun 2012.
21. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. Cryptology ePrint Archive, Report 2007/155, 2007. https://ia.cr/2007/155.
22. Lan Nguyen. Accumulators from bilinear pairings and applications. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, pages 275–292, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
23. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *Proceedings of the Cryptographers Track at the RSA Conference*, volume 9610 of *LNCS*, pages 111–126. Springer, 2016.
24. Olivier Sanders. Efficient redactable signature and application to anonymous credentials. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 628–656, Cham, 2020. Springer International Publishing.

25. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.
26. Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. *ArXiv*, abs/1802.07344, 2019.

A Proof of Lemma 1

Let $\mathcal{O}_{\text{Sign}}$ be an oracle parametrized with Schnorr public key $pk = (\mathbf{u}, \mathbf{v})$. Upon call with message m, $\mathcal{O}_{\text{Sign}}$ returns a valid Schnorr signature $\gamma = (\mathbf{f}, \pi)$.

Let $\mathcal{A}$ be an adversary whose goal is to produce a Schnorr signature forgery under public key pk.

Let $\mathcal{O}_{\text{RSign}}$ be an oracle parametrized with Schnorr public key $pk = (\mathbf{u}, \mathbf{v})$. Upon call with message m, $\mathcal{O}_{\text{RSign}}$ returns a tuple (pk', ξ', m', γ') such that

(1) $pk' = (\mathbf{u}', \mathbf{v}') \in \mathbb{G}_1 \times \mathbb{G}_1$;
(2) ξ' is a knowledge-extractable zero knowledge proof for binary relation $\mathcal{R}(pub, w)$ such that $pub = (pk, pk')$, $w = t' \in \mathbb{F}_q$ and $\mathcal{R}(pub, w) = 1$ iff $pk' = pk^{t'}$ (i.e., $(\mathbf{u}', \mathbf{v}') = (\mathbf{u}^{t'}, \mathbf{v}^{t'})$);
(3) γ' is a valid Schnorr signature of message $m'||\xi'$ under public key pk'.

Let $\mathcal{A}'$ be an adversary who produces a valid forgery of $\mathcal{O}_{\text{RSign}}$, that is, a tuple (pk', ξ', m', γ') that verifies (1), (2) and (3) and which was not output by $\mathcal{O}_{\text{RSign}}$.

For ease of exposition, we assume that the participants leverage ideal functionality $\mathcal{F}_{\text{NIZK}}$ (parameterized with relation $\mathcal{R}$) to generate the zero-knowledge proofs. This functionality is controlled by adversary $\mathcal{A}$.

To break Schnorr signature, $\mathcal{A}$ also simulates a random oracle $\mathcal{H}$ and the signing oracle $\mathcal{O}_{\text{RSign}}$.

Simulation of $\mathcal{H}$

Let H denote the cryptographic hash used in Schnorr signature under public key pk.

On input $(m'||\xi', pk', \mathbf{f}')$, query $\mathcal{F}_{\text{NIZK}}$ with verification request $(\text{VERIFY}, (pk, pk'), \xi')$.

- $\mathcal{F}_{\text{NIZK}}$ returns `false`: set $\mathcal{H}(m'||\xi', pk', \mathbf{f}')$ to $H(m'||\xi', pk', \mathbf{f}')$.
- $\mathcal{F}_{\text{NIZK}}$ returns `true`: using $\mathcal{F}_{\text{NIZK}}$ extract t' such that $pk' = pk^{t'}$ and output $H(m'||\xi', pk, \mathbf{f}'^{1/t'})$.

For arbitrary inputs, set the output of the random oracle to a randomly-selected value h.

Simulation of $\mathcal{O}_{\text{RSign}}$

On input of message m' from adversary $\mathcal{A}'$, adversary $\mathcal{A}$ proceeds as follows:

- compute $pk' = pk^{t'}$;
- call $\mathcal{F}_{\text{NIZK}}$ to generate proof ξ' for (pk, pk');
- call $\mathcal{O}_{\text{Sign}}$ with message $m = m'||\xi'$ to obtain signature $\gamma = (\mathbf{f}, \pi)$;
- return (pk', ξ', m', γ') where $\gamma' = (\mathbf{f}', \pi) = (\mathbf{f}^{t'}, \pi)$.

Note that thanks to the simulation of $\mathcal{H}$ the output of $\mathcal{A}'$ is indistinguishable from the output of signing oracle $\mathcal{O}_{\text{RSign}}$. In fact, $c = H(m, pk, \mathbf{f}) = H(m'||\xi', pk, \mathbf{f}'^{1/t'}) = \mathcal{H}(m'||\xi', pk', \mathbf{f}')$.

Also by construction: $\mathbf{u}^{\pi} = \mathbf{f}\mathbf{v}^{c}$. Hence:

$$
\begin{aligned}
(\mathbf{u}^{\pi})^{t'} &= (\mathbf{f}\mathbf{v}^{c})^{t'} \\
(\mathbf{u}^{t'})^{\pi} &= \mathbf{f}^{t'}(\mathbf{v}^{t'})^{c} \\
\mathbf{u}'^{\pi} &= \mathbf{f}'\mathbf{v}'^{c}
\end{aligned}
$$

This makes $\gamma' = (\mathbf{f}', \pi)$ a valid Schnorr signature of message $m'||\xi'$ relative to pk' and (pk', ξ', m', γ') a valid output of $\mathcal{O}_{\text{RSign}}$.

$\mathcal{A}$'s Forgery

Now $\mathcal{A}'$ outputs a forgery (pk', ξ', m', γ') that verifies

(1) $pk' = (\mathbf{u}', \mathbf{v}')$;
(2) ξ' is a valid knowledge-extractable zero knowledge proof for binary relation $\mathcal{R}$ relative to public input $pub = (pk, pk')$;
(3) $\gamma' = (\mathbf{f}', \pi')$ is a valid Schnorr signature of message $m'||\xi'$ under public key pk'.

Using $\mathcal{F}_{\text{NIZK}}$, adversary $\mathcal{A}$ first extracts t' such that $pk' = pk^{t'}$. Given t', $\mathcal{A}$ outputs pair (m, γ) where $m = m'||\xi'$ and $\gamma = (\mathbf{f}, \pi) = (\mathbf{f'}^{1/t'}, \pi')$.

Notice that, thanks to random oracle $\mathcal{H}$:

$$\begin{aligned} &\textsc{Schnorr.Verify}(m'||\xi', pk', \gamma') \rightarrow \texttt{true} \\ \implies &\textsc{Schnorr.Verify}(m, pk, \gamma) \rightarrow \texttt{true} \end{aligned}$$

This makes (m, γ) a valid forgery of Schnorr signature under public key pk, breaking hence the strong unforgeability of Schnorr signatures.

Notice that $\textsc{Schnorr.Verify}(m'||\xi', pk', \gamma') \rightarrow \texttt{true}$ entails:

$$\begin{aligned} \mathbf{u'}^{\pi'} &= \mathbf{f'}\mathbf{v'}^{c} \\ (\mathbf{u'}^{\pi'})^{1/t'} &= (\mathbf{f'}\mathbf{v'}^{c})^{1/t'} \\ \mathbf{u}^{\pi'} &= \mathbf{u}^{\pi} = \mathbf{f}\mathbf{v}^{c} \end{aligned}$$

Note also that

$$c = \mathcal{H}(m'||\xi', pk', \mathbf{f'}) = H(m'||\xi', pk, \mathbf{f'}^{1/t'}) = H(m, pk, \mathbf{f})$$

B $\mathcal{F}_{\text{NIZK}}$ Instantiation

We instantiate $\mathcal{F}_{\text{NIZK}}$ with Groth-Sahai (GS) proofs [21, 15, 17], which are commit-and-prove proof systems tailored for equations in bilinear groups, and which make use of GS commitments. These can be setup to either be perfectly hiding (ensuring witness indistinguishability) or perfectly binding (allowing extractability).

For completeness purposes, we introduce in the following, the definitions and notations relevant to GS proofs.

B.1 Groth-Sahai Commitments

We use Groth-Sahai commitments to commit to group elements – either in $\mathbb{G}_1$ or $\mathbb{G}_2$. For ease of exposition, we use $\mathbb{G}$ as a shorthand for both groups. We present here the instantiation that's secure under the Symmetric External Diffie-Hellman (SXDH) assumption.

Definition 2 (SXDH Assumption). *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three groups that admit an efficient bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.*

We say that SXDH assumption holds if DDH holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

SXDH-based Groth-Sahai commitments consist of the following algorithms.

- $\textsc{Keygen}(1^\kappa) \rightarrow ck$: on input of security parameter 1^κ, KEYGEN outputs a commitment public key $ck = (\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}) \in \mathbb{G}^4$.
- $\textsc{commit}(\mathbf{w}, r, s, ck) \rightarrow com$: on input of $\mathbf{w} \in \mathbb{G}$, randomness (r, s) and key $ck = (\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v})$, COMMIT outputs $com = (\mathbf{x}^r\mathbf{y}^s, \mathbf{w}\mathbf{u}^r\mathbf{v}^s)$. A commitment to a value $w \in \mathbb{F}_q$ is computed by calling COMMIT on input $\mathbf{w} = \mathbf{g}^w$.
- $\textsc{open}(\mathbf{w}, r, s, com, ck) \rightarrow \textsc{bool}$: on input of $\mathbf{w} \in \mathbb{G}$, randomness (r, s), commitment $com = (com_1, com_2)$ and key $ck = (\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v})$, OPEN returns `true` if $com_1 = \mathbf{x}^r\mathbf{y}^s$ and $com_2 = \mathbf{w}\mathbf{u}^r\mathbf{v}^s$, and `false` otherwise.

Note that ck can be generated so as the commitments are perfectly hiding. This is achieved by having generators $(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v})$ produced independently of each other. ck in this case is a *hiding key*. In the case where $(\mathbf{u}, \mathbf{v}) = (\mathbf{x}^t, \mathbf{y}^t)$ for some $t \in \mathbb{F}_q$, the commitment is perfectly binding and ck is a *binding key*.

Notice that when ck is binding, the commitment corresponds to an Elgamal encryption, and as a result, anyone with knowledge of the corresponding trapdoor $td = t$ can extract the committed group element.

B.2 Groth-Sahai Proofs

Groth-Sahai proofs are designed to prove statements expressed as equations over bilinear groups. In particular, they allow a prover to produce a NIZK for the following relations:

$$pub = (x_1, ..., x_m, \mathbf{y}_1, ..., \mathbf{y}_n, \mathbf{h})$$
$$\vec{w} = (w_1, ..., w_n, \mathbf{w}_1, ..., \mathbf{w}_m)$$
$$\mathcal{R}_{\mathbb{G}}(pub, \vec{w}) = 1 \iff \mathbf{h} = \prod_{i=1}^{m} \mathbf{w}_i^{x_i} \prod_{i=1}^{n} \mathbf{y}_i^{w_i}$$

and

$$pub = (\mathbf{g}_1, ..., \mathbf{g}_n, \mathbf{x}_1, ..., \mathbf{x}_m, \mathbf{y}_1, ..., \mathbf{y}_m)$$
$$\vec{\mathbf{w}} = (\mathbf{w}_1, ..., \mathbf{w}_n)$$
$$\mathcal{R}_{\mathbb{G}_t}(pub, \vec{\mathbf{w}}) = 1 \iff \prod_{i=1}^{n} e(\mathbf{g}_i, \mathbf{w}_i) \prod_{j=1}^{m} e(\mathbf{x}_j, \mathbf{y}_j) = 1$$

It is easy to see that $\mathcal{R}_{\mathbb{G}}$ and $\mathcal{R}_{\mathbb{G}_t}$ capture $\mathcal{R}$ depicted in Figure 3, and as a result, Groth-Sahai proofs can be used in a straightforward manner to instantiate $\mathcal{F}_{\text{NIZK}}$.

For completeness sake, we generically describe the algorithms underlying Groth-Sahai proofs.

- $\text{CRSGEN}(1^\kappa, \mathcal{R}) \to crs$: on input of security parameter 1^κ and relation $\mathcal{R}$, CRSGEN outputs a common reference string $crs = (ck, pp_{\mathcal{R}})$ whereby ck is the commitment key of Groth-Sahai commitments and $pp_{\mathcal{R}}$ are relation-dependent public parameters.
- $\text{COMMIT}(\vec{\mathbf{w}}, \vec{r}, \vec{s}, ck) \to \vec{com}$: on input of $\vec{\mathbf{w}} = (\mathbf{w}_1, ..., \mathbf{w}_n) \in \mathbb{G}^n$, vectors $\vec{r} = (r_1, ..., r_n) \in \mathbb{F}_q^n$ and $\vec{s} = (s_1, ..., s_n) \in \mathbb{F}_q^n$ and key ck, COMMIT outputs vector $\vec{com} = (com_1, ..., com_n)$ where $com_i \leftarrow \text{COMMIT}(\mathbf{w}_i, r_i, s_i, ck)$.
- $\text{PROVE}(crs, \vec{com}, \vec{\mathbf{w}}, \vec{r}, \vec{s}, pub) \to \xi$: on input of key ck, vectors $\vec{com}$, $\vec{\mathbf{w}}$, $\vec{r}$ and $\vec{s}$, and public input pub, PROVE produces a proof ξ.
- $\text{VERIFY}(crs, \vec{com}, pub, \xi) \to \text{BOOL}$: on input of crs, vector $\vec{com} = (com_1, ..., com_n)$, public input pub and proof ξ, VERIFY outputs `true` if ξ is valid; otherwise, it outputs `false`. ξ is valid if it proves knowledge of vectors $\vec{\mathbf{w}} = (\mathbf{w}_1, ..., \mathbf{w}_n)$, $\vec{r} = (r_1, ..., r_n)$ and $\vec{s} = (s_1, ..., s_n)$ such that:

$$\forall\ 1 \le i \le n, \text{OPEN}(\mathbf{w}_i, r_i, s_i, com_i, ck) \to \texttt{true}$$
$$\wedge\ \mathcal{R}(\vec{\mathbf{w}}, pub) = 1$$

- $\text{TDCRSGEN}(1^\kappa, \mathcal{R}) \to (crs, td)$: on input of security parameter 1^κ and relation $\mathcal{R}$, TDCRSGEN outputs $crs = (ck, pp_{\mathcal{R}})$ and a secret trapdoor td such that ck is a binding commitment key and td is the corresponding trapdoor.
- $\text{EXTRACT}(crs, td, \vec{com}, pub, \xi) \to \vec{\mathbf{w}}$: on input of crs, trapdoor td, vector $\vec{com} = (com_1, ..., com_n)$, public input pub and a valid proof ξ, EXTRACT outputs a vector $\vec{\mathbf{w}} = (\mathbf{w}_1, ..., \mathbf{w}_n) \in \mathbb{G}^n$ such that $\mathcal{R}(\vec{\mathbf{w}}, pub) = 1$.

Extractability Note that relation $\mathcal{R}$ depicted in Figure 3 mostly involves group elements – either in $\mathbb{G}_1$ or $\mathbb{G}_2$. The only exception is the proof of correct randomization of the user's public key that involves field element $t' \in \mathbb{F}_q$. As a result, the extraction of t would require commitments to the bit representation of t, which is not practical. Note that the extraction of t' is used in proof of Lemma 1 to compute $\mathbf{f} = \mathbf{f}'^{1/t'} = \mathbf{g}^f$ to both simulate random oracle $\mathcal{H}$ and compute $\mathcal{A}$'s forgery.

An alternative is to include in signature ψ an Elgamal ciphertext of $\mathbf{f}$ and enhance proof ξ to show that the ciphertext encrypts the correct information.

Let $\gamma' = (\mathbf{f}', \pi')$ be the Schnorr signature on $m||\xi$ with respect to randomized public key $upk' = (upk'_1, upk'_2)$.

We define relation $\widehat{\mathcal{R}}$ that augments relation $\mathcal{R}$ as follows: $\widehat{\mathcal{R}}(\widehat{pub}, \widehat{w}) = 1$ for $\widehat{w} = (w, f, s)$; $\widehat{pub} = (pub, \mathcal{C}, \mathbf{f}')$, **iff**:

$$\mathcal{R}(pub, w) = 1\ \wedge\ \mathbf{f}' = (upk'_1)^f$$
$$\wedge\ \mathcal{C} \leftarrow \text{ELGAMAL.ENC}(\mathbf{g}^f, s, ccrs)$$

The new proof allows the extraction of $\mathbf{f} = \mathbf{g}^f$ directly, which is all that is needed to successfully simulate random oracle $\mathcal{H}$.

Malleability Groth-Sahai proofs are malleable, however, the inclusion of proof ξ in the hash of Schnorr signature γ' ensures that ξ cannot be tampered with without detection (i.e., ξ becomes non-malleable).