Practical UC-Secure Delegatable Credentials with Attributes and Their Application to Blockchain

Jan Camenisch
IBM Research - Zurich
jca@zurich.ibm.com

Manu Drijvers
IBM Research - Zurich & ETH Zurich
mdr@zurich.ibm.com

Maria Dubovitskaya
IBM Research - Zurich
mdu@zurich.ibm.com

ABSTRACT

Certification of keys and attributes is in practice typically realized by a hierarchy of issuers. Revealing the full chain of issuers for certificate verification, however, can be a privacy issue since it can leak sensitive information about the issuer's organizational structure or about the certificate owner. Delegatable anonymous credentials solve this problem and allow one to hide the full delegation (issuance) chain, providing privacy during both delegation and presentation of certificates. However, the existing delegatable credentials schemes are not efficient enough for practical use.

In this paper, we present the first hierarchical (or delegatable) anonymous credential system that is practical. To this end, we provide a surprisingly simple ideal functionality for delegatable credentials and present a generic construction that we prove secure in the UC model. We then give a concrete instantiation using a recent pairing-based signature scheme by Groth and describe a number of optimizations and efficiency improvements that can be made when implementing our concrete scheme. The latter might be of independent interest for other pairing-based schemes as well. Finally, we report on an implementation of our scheme in the context of transaction authentication for blockchain, and provide concrete performance figures.

KEYWORDS

Credentials, Delegation, Hierarchical issuance, Privacy-preserving authentication, Composable Security, Zero-knowledge, Blockchain

1 INTRODUCTION

Privacy-preserving attribute-based credentials (PABCs) [6], originally introduced as anonymous credentials [10, 21], allow users to authenticate to service providers in a privacy-protecting way, only revealing the information absolutely necessary to complete a transaction. The growing legal demands for better protection of personal data and more generally the increasingly stronger security requirements make PABCs a primary ingredient for building secure and privacy-preserving IT systems.

CCS '17, October 30-November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134025

An *(attribute-based) anonymous credential* is a set of attributes certified to a user by an issuer. Every time a user presents her credential, she creates a fresh *token* which is a zero-knowledge proof of possession of a credential. When creating a token, the user can select which attributes she wants to disclose from the credential or choose to include only predicates on the attributes. Verification of a token requires knowledge of the issuer public key only. Despite their strong privacy features, anonymous credentials do reveal the identity of the issuer, which, depending on the use case, still leaks information about the user such as the user's location, organization, or business unit. In practice, credentials are typically issued in a hierarchical manner and thus the chain of issuers will reveal even more information. For instance, consider governmental issued certificates such as drivers licenses, which are typically issued by a local authority whose issuing keys are then certified by a regional authority, etc. So there is a hierarchy of at least two levels if not more. Thus, when a user presents her drivers license to prove her age, the local issuer's public key will reveal her place of residence, which, together with other attributes such as the user's age, might help to identify the user. As another example consider a (permissioned) blockchain. Such a system is run by multiple organizations that issue certificates (possibly containing attributes) to parties that are allowed to submit transactions. By the nature of blockchain, transactions are public or at least viewable by many blockchain members. Recorded transactions are often very sensitive, in particular when they pertain to financial or medical data and thus require protection, including the identity of the transaction originator. Again, issuing credential in a permissioned blockchain is a hierarchical process, typically consisting of two levels, a (possibly distributed) root authority, the first level consisting of CAs by the different organizations running the blockchain, and the second level being users who are allowed to submit transactions.

Delegatable anonymous credentials (DAC), formally introduced by Belenkiy et al. [3], can solve this problem. They allow the owner of a credential to *delegate* her credential to another user, who, in turn, can delegate it further as well as present it to a verifier for authentication purposes. Thereby, only the identity (or rather the public key) of the initial delegator (root issuer) is revealed for verification. A few DAC constructions have been proposed [3, 17, 20, 26], but none is suitable for practical use for the following reasons:

- While being efficient in a complexity theoretic sense, they are not practical because they use generic zero-knowledge proofs or Groth-Sahai proofs with many expensive pairing operations and a large credential size.
- The provided constructions are described mostly in a black-box fashion (to hide the complexity of their concrete instantiations), often leaving out the details that would be necessary for their implementation. Therefore, a substantial additional effort

would be required to translate these schemes to a software specification or perform a concrete efficiency analysis.
- The existing DAC security models do not consider attributes, which, however, are necessary in many practical applications. Also, extending the proposed schemes to include attributes on different delegation levels is not straightforward and will definitely not improve their efficiency.
- Finally, the existing schemes either do not provide an ideal functionality for DAC ([20]) or are proven secure in standalone models ([3, 17, 26]) that guarantee security only if a protocol is run in isolation, which is not the case for a real environment. In other words, no security guarantees are provided if they are used to build a system, i.e., the security of the overall system would have to be proved from scratch. This usually results in complex monolithic security proofs that are prone to mistakes and hard to verify.

The main reason why the existing schemes are sufficiently efficient, is that they hide the identities of the delegator and delegatee during credential delegation. Thus privacy is ensured for both delegation and presentation of credentials. While this is a superior privacy guarantee, we think that privacy is not necessary for delegation. Indeed, in real-world scenarios a delegator and a delegatee would typically know each other when an attribute-based credential is delegated, especially in the most common case of a hierarchal issuance. Therefore, we think that ensuring privacy only for presentation is a natural way to model delegatable credentials. Furthermore, revealing the full credential chain including the public keys and attribute values to the delegatee would allow us to avoid using expensive cryptographic building blocks such as generic zero-knowledge proofs, re-randomizable proofs, and malleable signatures.

1.1 Our Contribution

Let us look at delegatable credentials with a different privacy assumtions for delegation in mind and see how such system would work.

The root delegator (we call it *issuer*) generates a signing and a corresponding verification key and publishes the latter.

User A, to whom a credential gets issued on the first level (we call it a *Level-1 credential*), generates a fresh credential secret and a public key and sends the public key to the issuer. The issuer signs this public key together with the set of attributes and sends the generated signature to user A.

User A can then delegate her credential further to another user, say B, by signing B's freshly generated credential public key and (possibly another) set of attributes with the credential secret key of user A. A sends her signature together with her original credential and A's attributes to user B. User B's credential, therefore, consists of two signatures with the corresponding attribute sets, credential public keys of user A and user B, and B's credential secret key.

User B, using his credential secret key, can delegate his credential further as described above or use it to sign a message by generating a presentation token. The token is essentially a non-interactive zero-knowledge (NIZK) proof of possession of the signatures and the corresponding public keys from the delegation chain that does not reveal their values. The signed attributes can also be only selectively revealed using NIZK. Verification of the token requires only the public key of the issuer and, thus, hides the identities of both users A and B and (selectively) their attributes. Since all attributes, signatures, and public keys are revealed to the delegatee during delegation, we can use the most efficient zero-knowledge proofs (Schnorr proofs) that would make a protocol practical.

Contribution Summary. In this paper, we propose the first *practical* delegatable anonymous credential system with attributes that is well-suited for real-world applications.

More concretely, we first provide a (surprisingly simple) ideal functionality $\mathcal{F}_{\text{dac}}$ for delegatable credentials with attributes. Attributes can be different on any level of delegation. Each attribute at any level can be selectively revealed when generating presentation token. Tokens can be used to sign arbitrary messages. Privacy is guaranteed only during presentation, during delegation the delegatee knows the full credential chain delegated to her.

Second, we propose a generic DAC construction from signature schemes and zero-knowledge proofs and prove it secure in the universal composability (UC) framework introduced by Canetti [14]. Our construction can be used as a secure building block to build a higher-level system as a hybrid protocol, enabling a modular design and simpler security analysis.

Third, we describe a very efficient instantiation of our DAC scheme based on a recent structure-preserving signature scheme by Groth [29] and on Schnorr zero-knowledge proofs [34]. We further provide a thorough efficiency analysis of this instantiation and detailed pseudocode that can be easily translated into a computer program. We also discus a few optimization techniques for the type of zero-knowledge proofs we use (i.e., proofs of knowledge of group elements under pairings). These techniques are of independent interest.

Finally, we report on an implementation of our scheme in the context of a privacy-preserving membership service for permissioned blockchains and give concrete performance figures, demonstrating the practicality of our construction. For instance, generating an attribute token with four undisclosed attributes from a delegated credential takes only 50 miliseconds, and verification requires only 40 miliseconds, on a 3.1GHz Intel I7-5557U laptop CPU.

1.2 Related Work

There is only a handful of constructions of delegatable anonymous credentials [3, 17, 20, 26]. All of them provide privacy for both delegator and delegatee during credential delegation and presentation. The first one is by Chase and Lysyanskaya [20] which uses generic zero-knowledge proofs. The size of a credential in their scheme is exponential in the number of delegations, which, as authors admit themselves, makes it impractical and allows only for a constant number of delegations. Our ideal functionality for DAC is also quite different from the signature of knowledge functionality that they use to build a DAC system. For example, we distinguish between the delegation and presentation interfaces and ping the adversary for the delegation. We also do not require the extractability for the verification interface, which makes our scheme much more efficient.

The construction by Belenkiy et al. [3] employs Groth-Sahai NIZK proofs and in particular their randomization property. It allows for a polynomial number of delegations and requires a common reference string (CRS). Fuchsbauer [26] proposed a delegatable credential system that is inspired by the construction of Belenkiy et al. and supports non-interactive issuing and delegation of credentials. It is based on the commuting signatures and Groth-Sahai proofs and is at least twice as efficient as the scheme by Belenkiy et al. [3]. Our construction also requires a CRS, but still outperforms both schemes. For example, without attributes, the token size increases with every level by 4 group elements ($\mathbb{G}_1^2 \times \mathbb{G}_2^2$) for our scheme versus $\mathbb{G}_1^{50} \times \mathbb{G}_2^{40}$ for Belenkiy et al. [3] and $\mathbb{G}_1^{20} \times \mathbb{G}_2^{18}$ for Fuchsbauer [26]. Due to our optimization techniques, the number of expensive operations (exponentiations and pairings) is also minimized.

Finally, Chase et al. [17, 19] propose a DAC instantiation that is also non-interactive and scales linearly with the number of delegations. Their unforgeability definition is a bit different from the one by Belenkiy et al. [3] and implements the simulation extractability notion. However, none of the schemes accommodate attributes in their security definitions. As we mentioned above, it is hard to derive the exact efficiency figures from the "black-box"-type construction of [17], which is built from malleable signatures, which, in turn, are built from the malleable proofs. The efficiency of their scheme depends on the concrete instantiation of malleable proofs: either Groth-Sahai proofs [16], which would be in the same spirit as [26], or non-interactive arguments of knowledge (SNARKs) and homomorphic encryption [18], which, as the authors claim themselves, is less efficient.

Hierarchical group signatures, as introduced by Trolin and Wikström [35] and improved by Fuchsbauer and Pointcheval [27], are an extension of group signatures that allow for a tree of group managers. Users that received a credential from any of the managers can anonymously sign on behalf of the group, as is the case with delegatable credentials. However, in contrast to delegatable credentials, parties can serve either as manager or as user, but not both simultaneously. Additionally, hierarchical group signatures differ from delegatable credentials in the fact that signatures can be deanonymized by group managers.

2 PRELIMINARIES

This section introduces the notation and recalls well-known building blocks used in our delegatable credential scheme, such as signature schemes and zero-knowledge proofs. In addition, it defines a new primitive we call *sibling signatures*, that allow for two different signing algorithms sharing a single key pair. Finally, we give a brief overview of the universal composability framework.

2.1 Notation

Let $k \in \mathbb{N}$ denote the security parameter and $a \in \{0,1\}^*$ denote an input. Two binary distribution ensembles $X = \{X(k,a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$ and $Y = \{Y(k,a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$ are indistinguishable ($X \approx Y$) if for any $c, d \in \mathbb{N}$ there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$ and all $a \in \cup_{\kappa\leq k^d}\{0,1\}^\kappa$, $|\Pr[X(k,a)=1] - \Pr[Y(k,a)=1]| < k^{-c}$.

2.2 Bilinear Groups

Let $\mathcal{G}$ be a bilinear group generator that takes as an input a security parameter 1^κ and outputs the descriptions of multiplicative groups $\Lambda = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2)$ where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_t$ are groups of prime order q, e is an efficient, non-degenerating bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$, and g_1 and g_2 are generators of the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. We denote $\Lambda^* = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e)$ as Λ without group generators.

2.3 Zero-Knowledge Proofs

Feige, Fiat, and Shamir [24] were the first to formalize the proof of knowledge, while the concept of zero-knowledge was introduced by Goldwasser et al. [28]. When referring to the interactive proofs, one usually uses the notation introduced by Camenisch and Stadler [12] and formally defined by Camenisch, Kiayias, and Yung [9]. For instance, $\mathsf{PK}\{(a,b,c) : Y = g_1^a H^b \wedge \tilde{Y} = \tilde{g_1}^a \tilde{H}^c\}$ denotes a "*zero-knowledge Proof of Knowledge of integers a, b, c such that* $Y = g_1^a H^b$ *and* $\tilde{Y} = \tilde{g_1}^a \tilde{H}^c$ *holds*," where $y, g, h, \tilde{Y}, \tilde{g_1}$, and $\tilde{H}$ are elements of some groups $G = \langle g_1 \rangle = \langle H \rangle$ and $\tilde{G} = \langle \tilde{g_1} \rangle = \langle \tilde{H} \rangle$. The convention is that the letters in the parenthesis (a,b,c) denote quantities of which knowledge is being proven, while all other values are known to the verifier. $\mathsf{SPK}\{\ldots\}(m)$ denotes a signature proof of knowledge on m, which is a non-interactive transformation of such proofs using the Fiat-Shamir heuristic [25].

We can create similar proofs proving knowledge of group elements instead of exponents, e.g. $\mathsf{SPK}\{a \in \mathbb{G}_1 : y = e(a,b)\}$ by using $e(\cdot, b)$ instead of $b^{(\cdot)}$: Take $r \xleftarrow{\$} \mathbb{G}_1$, $t \leftarrow e(r,b)$, $c \leftarrow \mathsf{H}(\ldots) \in \mathbb{Z}_q$, and $s \leftarrow r \cdot a^c$. Verification computes $\hat{t} = e(s,b) \cdot y^{-c}$ and checks that the Fiat-Shamir hash [25] equals c. With the same mechanism we can prove knowledge of elements in $\mathbb{G}_2$.

We use $\mathsf{NIZK}\{w : s(w)\}$ to denote a generic non-interactive zero-knowledge proof proving knowledge of witness w such that statement $s(w)$ is true. Sometimes we need a witness to be online extractable by a simulator, which we denote by drawing a box around the witness: $\mathsf{NIZK}\{\boxed{w} : s(w)\}$.

2.4 Signature Schemes

A digital signature scheme Sig is a set of PPT algorithms $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$:

- $\mathsf{Sig.Setup}(1^\kappa) \xrightarrow{\$} sp$: The setup algorithm takes as input a security parameter and outputs public system parameters that also specify a message space $\mathcal{M}$.
- $\mathsf{Sig.Gen}(sp) \xrightarrow{\$} (sk, vk)$: The key generation algorithm takes as input system parameters and outputs a verification key vk and a corresponding secret key sk.
- $\mathsf{Sig.Sign}(sk, m) \xrightarrow{\$} \sigma$: The signing algorithm takes as input a private key sk and a message $m \in \mathcal{M}$ and outputs a signature σ.
- $\mathsf{Sig.Verify}(vk, m, \sigma) \to 1/0$: The verification algorithm takes as input a public verification key vk, a message m and a signature σ and outputs 1 for acceptance or 0 for rejection according to the input.

Our generic construction is built from the structure-preserving [1] signature schemes. A signature scheme Sig over a bilinear

group Λ generated by $\mathcal{G}(1^\kappa)$, that outputs system parameters $\Lambda = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2)$, is said to be structure preserving if:

(1) the verification key vk consists of the group parameters and group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$;
(2) the messages and the signatures consist of group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$, and
(3) the verification algorithm evaluates membership $\mathbb{G}_1$ and $\mathbb{G}_2$ and pairing product equations of the form

$$\prod_i \prod_j e(g_i, h_j)^{a_{ij}} = 1_{\mathbb{G}_t},$$

where $a_{11}, a_{12}, \ldots \in \mathbb{Z}_q$ are constants, $g_1, h_1, \ldots \in \{\mathbb{G}_1, \mathbb{G}_2\}^*$ are group elements appearing in the group parameters, verification key, messages, and signatures.

2.4.1 Structure-Preserving Signature scheme by Groth (Asiacrypt 2015). We recall the structure-preserving signature scheme by Groth [29], which we use in our instantiation and refer to as Groth. We note that the original scheme supports signing blocks of messages in a form of "matrix", whereas we provide a simplified description for "vectors" of messages only, since we use this version of the signature scheme in our construction. Let a message be a vector of group elements of length n: $\vec{m} = (m_1, \ldots, m_n)$. We denote as Groth1 signs messages in $\mathbb{G}_1$ with a public key in $\mathbb{G}_2$, while Groth2 signs messages in $\mathbb{G}_2$ with a public key in $\mathbb{G}_1$. We describe the Groth2 scheme below. Groth1 follows immediately.

Groth2.Setup: Let $\Lambda^* = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e)$ and $y_i \xleftarrow{\$} \mathbb{G}_2$ for $i = 1, \ldots, n$. Output parameters $sp = (\Lambda^*, \{y_i\}_{i=1,\ldots,n})$.

Groth2.Gen(sp): Choose random $v \xleftarrow{\$} \mathbb{Z}_q$ and set $V \xleftarrow{\$} g_1^v$. Output public verification key $vk = V$ and secret key $sk = v$.

Groth2.Sign($sk; \vec{m}$): To sign message $\vec{m} \in \mathbb{G}_2^n$ choose a random $r \xleftarrow{\$} \mathbb{Z}_q^*$ and set

$$R \leftarrow g_1^r \quad S \leftarrow (y_1 \cdot g_2^v)^{\frac{1}{r}} \quad T_i \leftarrow (y_i^v \cdot m_i)^{\frac{1}{r}}.$$

Output signature $\sigma = (R, S, T_1, \ldots, T_n)$.

Groth2.Verify($vk, \sigma, \vec{m}$) On input message $\vec{m} \in \mathbb{G}_2^n$ and signature $\sigma = (R, S, T_1, \ldots, T_n) \in \mathbb{G}_1 \times \mathbb{G}_2^{n+1}$, output 1 iff

$$e(R, S) = e(g_1, y_1)e(V, g_2) \wedge \bigwedge_{i=1}^{n} e(R, T_i) = e(V, y_i)e(g_1, m_i)$$

Groth2.Rand(σ) To randomize signature $\sigma = (R, S, T_1, \ldots, T_n)$, pick $r' \xleftarrow{\$} \mathbb{Z}_q$ and set

$$R' \leftarrow R^{r'} \quad S' \leftarrow S^{\frac{1}{r'}} \quad T_i' \leftarrow T_i^{\frac{1}{r'}}.$$

Output randomized signature $\sigma' = (R', S', T_1', \ldots, T_n')$.

2.5 Sibling Signatures

We introduce a new type of signatures that we call *sibling signatures*. It allows a signer with one key pair to use two different signing algorithms, each with a dedicated verification algorithm. In our generic construction, this will allow a user to hold a single key pair that it can use for both presentation and delegation of a credential.

A sibling signature scheme consists of algorithms Setup, Gen, Sign_1, Sign_2, Verify_1, Verify_2.

$\mathsf{Sib.Setup}(1^\kappa) \xrightarrow{\$} sp$: The setup algorithm takes as input a security parameter and outputs public system parameters that also specify two message spaces $\mathcal{M}_1$ and $\mathcal{M}_2$.

$\mathsf{Sib.Gen}(sp) \xrightarrow{\$} (sk, vk)$: The key generation algorithm takes as input system parameters and outputs a verification key vk and a corresponding secret key sk.

$\mathsf{Sib.Sign}_1(sk, m) \xrightarrow{\$} \sigma$: The signing algorithm takes as input a private key sk and a message $m \in \mathcal{M}_1$ and outputs a signature σ.

$\mathsf{Sib.Sign}_2(sk, m) \xrightarrow{\$} \sigma$: The signing algorithm takes as input a private key sk and a message $m \in \mathcal{M}_2$ and outputs a signature σ.

$\mathsf{Sib.Verify}_1(vk, m, \sigma) \rightarrow 1/0$: The verification algorithm takes as input a public verification key vk, a message m and a signature σ and outputs 1 for acceptance or 0 for rejection according to the input.

$\mathsf{Sib.Verify}_2(vk, m, \sigma) \rightarrow 1/0$: The verification algorithm takes as input a public verification key vk, a message m and a signature σ and outputs 1 for acceptance or 0 for rejection according to the input.

We require sibling signatures to be *complete* and *unforgeable*.

Definition 2.1 (Completeness). A sibling signature scheme is complete if for $b \in \{0, 1\}$ and for all $m \in \mathcal{M}_b$ we have

$$\Pr\left[\mathsf{Sib.Verify_b}(vk, m, \sigma) = 1 | sp \xleftarrow{\$} \mathsf{Sib.Setup}(1^\kappa), (sk, vk) \xleftarrow{\$} \mathsf{Sib.Gen}(sp), \sigma \xleftarrow{\$} \mathsf{Sib.Sign_b}(sk, m)\right] = 1.$$

Definition 2.2 (Unforgeability). No adversary with oracle access to Sign_1 and Sign_2 can create a signature that correctly verifies with $\mathsf{Verify_b}$, if no $\mathsf{Sign_b}$ query was made for message m. For every such $b \in \{1, 2\}$ we call it *unforgeability-b*. More precisely, a sibling signature scheme is unforgeable-b if the probability

$$\Pr\left[\mathsf{Sib.Verify_b}(vk, m, \sigma) = 1 \wedge m \notin Q_{\mathsf{Sign_b}} | sp \xleftarrow{\$} \mathsf{Sib.Setup}(1^\kappa), (sk, vk) \xleftarrow{\$} \mathsf{Sib.Gen}(sp), (\sigma, m) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sib.Sign}_1(sk,\cdot)}, \mathcal{O}^{\mathsf{Sib.Sign}_2(sk,\cdot)}}(sp, vk)\right]$$

is negligible in κ for every PPT adversary $\mathcal{A}$ and $b \in \{1, 2\}$, where oracle $\mathcal{O}^{\mathsf{Sib.Sign_b}(sk,\cdot)}$ on input m stores m in $Q_{\mathsf{Sign_b}}$ and returns $\mathsf{Sib.Sign_b}(sk, m)$. A sibling signature scheme is unforgeable if it is both unforgeable-1 and unforgeable-2.

2.5.1 Constructing Sibling Signatures. Note that one can trivially construct a sibling signature scheme from two standard signature schemes by setting the verification key vk as (vk_1, vk_2) and the signing key as $sk = (sk_1, sk_2)$, and simply using one signature scheme as Sign_1 and Verify_1 and the other as Sign_2 and Verify_2. However, this generalization also allows for instantiations that securely share key material between the two algorithms.

We now show that one can combine Groth1 signatures with Schnorr-signatures to form a sibling signature scheme we call SibGS1. SibGS1 uses only a single key pair. It uses the Setup and Gen algorithms of Groth1. Algorithm Sign_1 is instantiated with Groth1.Sign, and Sign_2 creates a Schnorr signature. Let SibGS2 denote the analogously defined Groth-Schnorr sibling signature where we use Groth2 instead of Groth1.

Lemma 2.3. SibGSb *is a secure sibling signature scheme in the random oracle and generic group model.*

Proof. Completeness of SibGSb directly follows from the completeness of Grothb and Schnorr signatures. We can reduce the unforgeability-1 and unforgeability-2 of SibGSb to the unforgeability of Grothb, which is proven to be unforgeable in the generic group model. The reduction algorithm $\mathcal{B}$ receives the Grothb verification key vk from the challenger and has access to signing oracle $\mathcal{O}^{\text{Grothb.Sign}(sk,\cdot)}$ that creates signatures valid under vk. $\mathcal{B}$ simulates the random oracle honestly and must answer $\mathcal{A}$'s signing queries by simulating oracles $\mathcal{O}^{\text{Sib.Sign}_1(sk,\cdot)}$ and $\mathcal{O}^{\text{Sib.Sign}_2(sk,\cdot)}$. When $\mathcal{A}$ queries $\mathcal{O}^{\text{Sib.Sign}_1(sk,\cdot)}$ on m, $\mathcal{B}$ queries $\sigma \leftarrow \mathcal{O}^{\text{Grothb.Sign}(sk,\mathbb{m})}$ and returns σ. When $\mathcal{A}$ queries $\mathcal{O}^{\text{Sib.Sign}_2(sk,\cdot)}$ on m, $\mathcal{B}$ simulates a Schnorr signature without knowledge of sk by programming the random oracle.

Finally, $\mathcal{A}$ outputs a forgery. Let us first consider the unforgeability-1 game, meaning that $\mathcal{A}$ outputs forgery σ^* on message m^*, such that σ^* is a valid Grothb signature on m^* and $\mathcal{O}^{\text{Sib.Sign}_1(sk,\cdot)}$ was not queried on m^*. This means that $\mathcal{B}$ did not query $\mathcal{O}^{\text{Grothb.Sign}(sk,\cdot)}$ on m^*, so $\mathcal{B}$ can break the unforgeability of Grothb by submitting forgery (σ^*, m^*).

Next, consider the unforgeability-2 game. Forgery σ^* is a Schnorr signature on m^* and $\mathcal{A}$ did not query $\mathcal{O}^{\text{Sib.Sign}_1(sk,\cdot)}$ on m^*. This means that the Schnorr signature is not a simulated signature and we use the forking lemma [4] to extract sk. Now, $\mathcal{B}$ picks a new message $\hat{m}^*$ for which it did not query $\mathcal{O}^{\text{Grothb.Sign}(sk,\cdot)}$, and uses sk to create signature $\hat{\sigma}^*$ on $\hat{m}^*$. It submits $(\hat{\sigma}^*, \hat{m}^*)$ as its forgery to win the Grothb unforgeability game. □

2.6 Universal Composability

We define the security of delegatable credentials as an ideal functionality in the Universal Composability (UC) framework [14], which follows the simulation-based security paradigm [8, 14, 30, 31, 33]. In UC, an environment $\mathcal{E}$ gives input to the protocol participants and receives their outputs. In the real world, honest parties execute the protocol over a network controlled by an adversary $\mathcal{A}$, who also controls the corrupt parties while communicating freely with environment $\mathcal{E}$. In the ideal world, honest parties are "dummy parties" who forward their inputs to the ideal functionality $\mathcal{F}$. The ideal functionality internally performs the desired task and generates outputs for honest parties.

Informally, a protocol Π securely realizes an ideal functionality $\mathcal{F}$ if the real world is as secure as the ideal world. For every adversary $\mathcal{A}$ attacking the real world, there exists a simulator $\mathcal{S}$ that performs an equivalent attack on the ideal world. As $\mathcal{F}$ performs the task at hand in an ideal fashion ($\mathcal{F}$ is secure by construction) there are no meaningful attacks on the ideal world, it follows that there are no meaningful attacks on the real world. More precisely, Π securely realizes $\mathcal{F}$ if for every PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that no PPT environment $\mathcal{E}$ can distinguish the real world (with Π and $\mathcal{A}$) from the ideal world (with $\mathcal{F}$ and $\mathcal{S}$).

The UC framework comes with a composability theorem, which gives composability guarantees for protocols proven secure in this framework. Specifically, it proves that security is preserved while running many instances of the same protocol in paralel, and when using the protocol as a building block for more advanced protocols.

We consider only static corruptions in this paper.

We now formally define the ideal functionalities that we use in our protocol.

Ideal Functionality $\mathcal{F}_{\text{crs}}$. For the CRS functionality we use the 2005 version of UC [13]. Functionality $\mathcal{F}_{\text{crs}}$ is parametrized by a distribution $\mathcal{D}$, from which the CRS is sampled.

Functionality $\mathcal{F}_{\text{crs}}$

(1) When receiving input (CRS, *sid*) from party $\mathcal{P}$, first verify that $sid = (\{\mathcal{P}\}, sid')$ where $\{\mathcal{P}\}$ is a set of identities, and that $\mathcal{P} \in \{\mathcal{P}\}$; else ignore the input. Next, if there is no value r recorded then choose and record $r \xleftarrow{\$} \mathcal{D}$. Finally, send a public delayed output (CRS, *sid*, r) to $\mathcal{P}$.

Ideal Functionality $\mathcal{F}_{\text{ca}}$. We use the ideal certification authority functionality $\mathcal{F}_{\text{ca}}$ as defined in [15].

Functionality $\mathcal{F}_{\text{ca}}$

(1) Upon receiving the first message (REGISTER, *sid*, v) from party $\mathcal{P}$, send (REGISTERED, *sid*, v) to the adversary $\mathcal{A}$; upon receiving OK from $\mathcal{A}$, and if $sid = \mathcal{P}$ and this is the first request from $\mathcal{P}$, then record the pair $(\mathcal{P}, v)$.
(2) Upon receiving a message (RETRIEVE, *sid*) from party $\mathcal{P}'$, send (RETRIEVE, *sid*, $\mathcal{P}'$) to $\mathcal{A}$, and wait for an OK from $\mathcal{A}$. Then, if there is a recorded pair (sid, v) output (RETRIEVE, *sid*, v) to $\mathcal{P}'$. Else output (RETRIEVE, *sid*, $\perp$) to $\mathcal{P}'$.

Ideal Functionality $\mathcal{F}_{\text{smt}}$. We use the secure message transmission functionality as defined in the 2005 version of UC [13]. Functionality $\mathcal{F}_{\text{smt}}^{l}$ is parameterized by a leakage function $l : \{0,1\}^* \rightarrow \{0,1\}^*$ that leaks information about the transmitted message, for example a message length.

Functionality $\mathcal{F}_{\text{smt}}$

(1) On input (SEND, *sid*, m) from a party $\mathcal{P}$, abort if $sid \neq (\mathcal{S}, \mathcal{R}, sid')$, send (SEND, *sid*, $l(m)$) to the adversary, generate a private delayed output (SENT, *sid*, m) to $\mathcal{R}$ and halt.

3 DEFINITION OF DELEGATABLE CREDENTIALS

We now define delegatable credentials in the form of an ideal functionality $\mathcal{F}_{\text{dac}}$. For simplicity we consider the functionality with a single root delegator (issuer), but using multiple instances of $\mathcal{F}_{\text{dac}}$ allows for many issuers. $\mathcal{F}_{\text{dac}}$ allows for multiple levels delegation. A Level-1 credential is issued directly by the issuer. Any further

(1) `Setup`. On input (SETUP, sid, $\langle n_i \rangle_i$) from $\mathcal{I}$.
- Verify that $sid = (\mathcal{I}, sid')$.
- Output (SETUP, sid, $\langle n_i \rangle_i$) to $\mathcal{A}$ and wait for response (SETUP, sid, Present, Ver, $\langle \mathbb{A}_i \rangle_i$) from $\mathcal{A}$.
- Store algorithms Present and Ver and credential parameters $\langle \mathbb{A}_i \rangle_i, \langle n_i \rangle_i$, initialize $\mathcal{L}_{\mathsf{de}} \leftarrow \emptyset$; $\mathcal{L}_{\mathsf{at}} \leftarrow \emptyset$.
- Output (SETUPDONE, sid) to $\mathcal{I}$.

(2) `Delegate`. On input (DELEGATE, sid, $ssid$, $\vec{a}_1, \ldots, \vec{a}_L, \mathcal{P}_j$) from some party $\mathcal{P}_i$, with $\vec{a}_L \in \mathbb{A}_L^{n_L}$.
- If $L = 1$, check $sid = (\mathcal{P}_i, sid')$ and add an entry $\langle \mathcal{P}_j, \vec{a}_1 \rangle$ to $\mathcal{L}_{\mathsf{de}}$.
- If $L > 1$, check that an entry $\langle \mathcal{P}_i, \vec{a}_1, \ldots, \vec{a_{L-1}} \rangle$ exists in $\mathcal{L}_{\mathsf{de}}$.
- Output (ALLOWDEL, sid, $ssid$, $\mathcal{P}_i$, $\mathcal{P}_j$, L) to $\mathcal{A}$ and wait for input (ALLOWDEL, sid, $ssid$) from $\mathcal{A}$.
- Add an entry $\langle \mathcal{P}_j, \vec{a}_1, \ldots, \vec{a_L} \rangle$ to $\mathcal{L}_{\mathsf{de}}$.
- Output (DELEGATE, sid, $ssid$, $\vec{a}_1, \ldots, \vec{a_L}, \mathcal{P}_i$) to $\mathcal{P}_j$.

(3) `Present`. On input (PRESENT, sid, m, $\vec{a}_1, \ldots, \vec{a}_L$) from some party $\mathcal{P}_i$, with $\vec{a}_i \in (\mathbb{A}_i \cup \perp)^{n_i}$ for $i = 1, \ldots, L$.
- Check that an entry $\langle \mathcal{P}_i, \vec{a}_1', \ldots, \vec{a}_L' \rangle$ exists in $\mathcal{L}_{\mathsf{de}}$ such that $\vec{a}_i \preceq \vec{a}_i{}'$ for $i = 1, \ldots, L$.
- Set $at \leftarrow \mathsf{Present}(m, \vec{a}_1, \ldots, \vec{a}_L)$ and abort if $\mathsf{Ver}(at, m, \vec{a}_1, \ldots, \vec{a}_L) = 0$.
- Store $\langle m, \vec{a}_1, \ldots, \vec{a}_L \rangle$ in $\mathcal{L}_{\mathsf{at}}$.
- Output (TOKEN, sid, at) to $\mathcal{P}_i$.

(4) `Verify`. On input (VERIFY, sid, at, m, $\vec{a}_1, \ldots, \vec{a}_L$) from some party $\mathcal{P}_i$.
- If there is no record $\langle m, \vec{a}_1, \ldots, \vec{a}_L \rangle$ in $\mathcal{L}_{\mathsf{at}}$, $\mathcal{I}$ is honest, and for $i = 1, \ldots, L$, there is no corrupt $\mathcal{P}_j$ such that $\langle \mathcal{P}_j, \vec{a}_1', \ldots, \vec{a}_i' \rangle \in \mathcal{L}_{\mathsf{de}}$ with $\vec{a}_j \preceq \vec{a}_j'$ for $j = 1, \ldots, i$, set $f \leftarrow 0$.
- Else, set $f \leftarrow \mathsf{Ver}(at, m, \vec{a}_1, \ldots, \vec{a}_L)$.
- Output (VERIFIED, sid, f) to $\mathcal{P}_i$.

Figure 1: Ideal functionality for delegatable credentials with attributes $\mathcal{F}_{\mathsf{dac}}$

delegations are done between users: the owner of a Level-$(L-1)$ credential can delegate it further, giving the receiver a Level L credential. $\mathcal{F}_{\mathsf{dac}}$ supports attributes on every level; attributes can be selectively disclosed during credential presentation. A presentation of a delegated credential creates a so-called *attribute token*, which can be verified with respect to the identity of the issuer, hiding the identity of the delegators.

$\mathcal{F}_{\mathsf{dac}}$ interacts with the issuer $\mathcal{I}$ and parties $\mathcal{P}_i$ who can delegate, present, and verify the credentials through the following four interfaces: SETUP, DELEGATE, PRESENT, VERIFY, that we describe here. The formal definition is presented in Fig. 1, where we use two conventions that ease the notation. First, the SETUP interface can only be called once, and all other interfaces ignore all input until a SETUP message has been completed. Second, whenever $\mathcal{F}_{\mathsf{dac}}$ performs a check, it means that if the check fails, it aborts by outputting $\perp$ to the caller.

Setup. The SETUP message is sent by the issuer $\mathcal{I}$, whose identity is fixed in the session identifier sid: $\mathcal{F}_{\mathsf{dac}}$ first checks that $sid = (\mathcal{I}, sid')$, which guarantees that each issuer can initialize its own instance of the functionality. The issuer defines the number of attributes for every delegation level i by specifying $\langle n_i \rangle_i$. This can be done efficiently by describing a function $f(i)$. We fix the number of attributes on the same delegation level since different number of attributes used by different delegators on the same level may leak information about the delegators. $\mathcal{I}$ does not need to specify the maximum number of the delegation levels.

$\mathcal{F}_{\mathsf{dac}}$ then asks the adversary for algorithms and credential parameters. The adversary provides algorithms Present, Ver for presenting and verifying attribute tokens, respectively, and specifies the attribute spaces $\langle \mathbb{A}_i \rangle_i$ for different credential levels. $\mathcal{F}_{\mathsf{dac}}$ stores Present, Ver, $\langle \mathbb{A}_i \rangle_i$, $\langle n_i \rangle_i$ and initializes two empty sets: $\mathcal{L}_{\mathsf{de}}$ for delegation and $\mathcal{L}_{\mathsf{at}}$ for presentation bookkeeping.

Delegate. The DELEGATE message is sent by a user $\mathcal{P}_i$ with a Level-$(L-1)$ credential to delegate it to a user $\mathcal{P}_j$, giving $\mathcal{P}_j$ a Level-L credential. $\mathcal{P}_i$ specifies a list of attribute vectors for all the previous levels in the delegation chain $\vec{a}_1, \ldots, \vec{a}_{L-1}$ and the vector of attributes $\vec{a}_L$ to certify in a freshly delegated Level-L credential. All attribute vectors should satisfy the corresponding attribute space and length requirements. We use subsession identifiers in this interface since multiple delegation sessions might be interleaved due to the communication with the adversary. If this delegation gives $\mathcal{P}_j$ a Level-1 credential, then $\mathcal{F}_{\mathsf{dac}}$ verifies that party $\mathcal{P}_i$ is the issuer by checking the sid and adds an entry $\langle \mathcal{P}_j, \vec{a}_1 \rangle$ to $\mathcal{L}_{\mathsf{de}}$. If this is not the first level delegation ($L > 1$), $\mathcal{F}_{\mathsf{dac}}$ checks if $\mathcal{P}_i$ indeed has a Level-$(L-1)$ credential with the specified attributes $\vec{a}_1, \ldots, \vec{a}_{L-1}$ by looking it up in $\mathcal{L}_{\mathsf{de}}$. $\mathcal{F}_{\mathsf{dac}}$ then asks the adversary if the delegation should proceed and, after receiving a response from $\mathcal{A}$, adds the corresponding delegation record to $\mathcal{L}_{\mathsf{de}}$ and sends the output that includes the full attribute chain to $\mathcal{P}_j$, notifying it of the successful delegation.

Note that in contrast to previous work on delegatable credentials, we model no privacy in delegation. That is, $\mathcal{P}_i$ and $\mathcal{P}_j$ will learn the identity of eachother during delegation. While this is a weaker privacy definition than previous definitions, we think privacy for delegation is not neccesary. in real-world scenarios, the delegator and delegatee will typically know eachother when a credential with attributes is delegated.

Present. The PRESENT message is sent by a user $\mathcal{P}_i$ to create an attribute token. A token selectively reveals attributes from the delegated credential and also signs a message m, which can be an arbitrary string. $\mathcal{P}_i$ inputs attribute vectors by specifying only the values of the disclosed attributes and using special symbol $\perp$ to indicate the hidden attributes. $\mathcal{F}_{\mathsf{dac}}$ checks if a delegation entry exists in $\mathcal{L}_{\mathsf{de}}$ such that the corresponding disclosed attributes were indeed delegated to $\mathcal{P}_i$. For this, it uses the following relation for

attribute vectors: We say that for two vectors $\vec{a} = (a_1, \ldots, a_n)$; $\vec{b} = (b_1, \ldots, b_n) : \vec{a} \preceq \vec{b}$ if $a_i = b_i$ or $a_i = \perp$ for $i = 1, \ldots, n$.

If this is the case it runs the Present algorithm to generate the attribute token. The Present algorithm does not take the identity of the user and the non-disclosed attributes as input - the attribute token is computed independently of these values. This ensures the user's privacy and hiding the non-disclosed attributes on all levels of the delegated credential chain. Next, it checks that the computed attribute token is valid using the Ver algorithm, which ensures completeness. It outputs the token value to user $\mathcal{P}_i$.

Verify. The VERIFY message is sent by a user $\mathcal{P}_i$ to verify an attribute token. Message m and the disclosed attribute values are also provided as input for verification. $\mathcal{F}_{\text{dac}}$ performs the unforgeability check: if the message together with the corresponding disclosed attribute values were not signed by calling the PRESENT interface (there is no corresponding bookkeeping record), the issuer is honest, and on any delegation level there is no corrupted party with the matching attributes, then $\mathcal{F}_{\text{dac}}$ outputs a negative verification result; otherwise, $\mathcal{F}_{\text{dac}}$ runs the verification algorithm and outputs the result to $\mathcal{P}_i$.

Our ideal functionality $\mathcal{F}_{\text{dac}}$ can be easily extended to also accept as input and output commitments to attribute values, following the recent work by Camenisch et al. [7], which would allow extending our delegatable credential scheme with existing revocation schemes for anonymous credentials in a hybrid protocol.

4 A GENERIC CONSTRUCTION FOR DELEGATABLE CREDENTIALS

In this section, we provide a generic construction for delegatable anonymous credentials with attributes. We first explain the intuition behind our construction, then present a construction based on sibling signatures defined in Section 2.5 and non-interactive zero-knowledge proofs. Then we prove that our generic construction securely realizes $\mathcal{F}_{\text{dac}}$. We provide an efficient instantiation of our generic construction in the next section.

4.1 Construction Overview

Recall that our definition of delegatable credentials allows for multiple levels of delegation. There is a root delegator (also called issuer) that issues Level-1 credentials to users. Users can delegate their Level-L credential, resulting in a Level-$(L+1)$ credential. We now explain on a high level how a user obtains a Level-1 credential and then that credential is delegated. It is then easy to see how a Level-L credential is delegated (this is also depicted in Fig. 2).

The issuer first generates a signing key *isk* and corresponding verification key *ipk* and publishes *ipk*, after which it can issue a Level-1 credential to a user. The user, to get Level-1 credential issued, generates a fresh secret and a public key (csk_1, cpk_1) for this credential and sends public key cpk_1 to the root delegator. The root delegator signs this public key together with a set of attributes $\vec{a}_1$ and sends the signature σ_1 back to the user. A Level-1 credential $cred_1$ consists of the signature σ_1, attributes $\vec{a}_1$, and credential keys (cpk_1, csk_1).

The user can delegate $cred_1$ further to another user by issuing a Level-2 credential. The receiver generates a fresh key pair (csk_2, cpk_2) for the Level-2 credential. The delegation is done by signing public key cpk_2 and a set of attributes $\vec{a}_2$ (chosen by the delegator) with the Level-1 credential secret key csk_1. The resulting signature σ_2 is sent back together with the attributes $\vec{a}_2$ and the original signature σ_1, and the corresponding attributes $\vec{a}_1$. The Level-2 credential consists of both signatures σ_1, σ_2, attributes $\vec{a}_1, \vec{a}_2$, and keys cpk_1, cpk_2, csk_2. Note that the Level-2 credential is a chain of two so-called *credential links*. The first link, consisting of $(\sigma_1, \vec{a}_1, cpk_1)$ proves that the delegator has a Level-1 credential containing attributes $\vec{a}_1$. The second link, $(\sigma_2, \vec{a}_2, cpk_2)$, proves that this delegator issued attributes $\vec{a}_2$ to the owner of cpk_2. The key csk_2 allows the user to prove he is the owner of this Level-2 credential. Note, that the Level-1 credential secret key csk_1 is not sent together with the signature and the credential link, so that it is impossible for a user who owns the Level-2 credential to present or delegate the Level-1 credential.

The Level-2 credential can be delegated further in the analogous way by generating a signature on attributes and a public key and sending them together with lower-level credential links. A Level-L credential is therefore a chain of the L credential links, where every link adds a number of attributes $\vec{a}_i$, and a secret key csk_L that allows the owner to present the credential or to delegate it further.

A credential of any level can be presented by its owner by generating a NIZK proof proving a possession of all credential links back to the issuer and selectively disclosing attributes from the corresponding signatures. This proof, that we call an *attribute token*, can be verified with the public key of only the issuer. The public keys of all the credential links remain hidden in the zero-knowledge proof and, therefore, the identities of all the intermediate delegators are not revealed by the attribute token.

4.2 Generic Construction

Our generic construction Π_{dac} is based on secure sibling signature schemes, where Sign_1 signs vectors of messages. We allow different

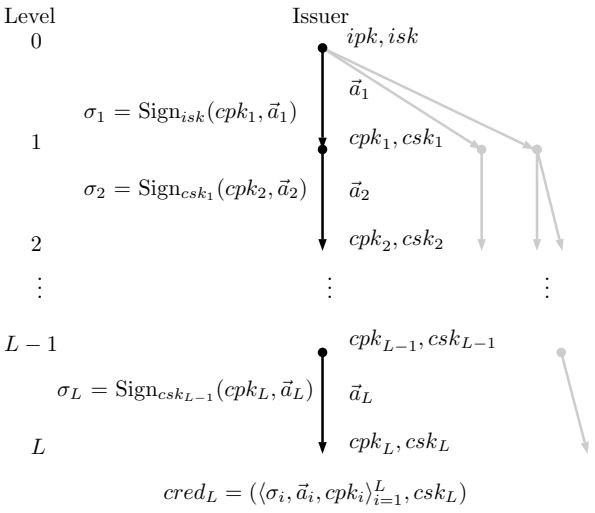

Figure 2: Our Generic Construction: Delegation

sibling signature schemes to be used at different delegation levels. Let Sib_i denote the scheme used by the owners of Level-i credentials. As we sign public keys of another signature scheme and the attribute values, the different signature schemes must be compatible with each other: The public key space of Sib_{i+1} must be included in the message space $\mathcal{M}_1$ of Sib_i. It follows that the attribute space $\mathbb{A}_i$ is the message space of Sib_{i-1}.

The required system parameters for the signature schemes are taken from $\mathcal{F}_{\mathsf{crs}}$. We implicitly assume that every protocol participant queries $\mathcal{F}_{\mathsf{ca}}$ to retrieve the issuer public key and $\mathcal{F}_{\mathsf{crs}}$ to retrieve the system parameters, and that the system parameters are passed as an implicit input to every algorithm of the signature schemes.

Setup. In the setup phase, the issuer $\mathcal{I}$ creates his key pair and registers this with the CA functionality $\mathcal{F}_{\mathsf{ca}}$.

(1) $\mathcal{I}$, upon receiving input (SETUP, *sid*, $\langle n_i \rangle_i$):
- Check that $sid = \mathcal{I}, sid'$ for some sid'.
- Run $(ipk, isk) \leftarrow \mathsf{Sib}_0.\mathsf{Gen}(1^\kappa)$ and compute proof $\pi_{isk} \leftarrow \mathsf{NIZK}\{\boxed{isk} : (ipk, isk) \in \mathsf{Sib}_0.\mathsf{Gen}(1^\kappa)\}$. Register public key (ipk, π_{isk}) with $\mathcal{F}_{\mathsf{ca}}$. Let $cpk_0 \leftarrow ipk$.
- Output (SETUPDONE, *sid*).

Delegate. Any user $\mathcal{P}_i$ with a Level-$L-1$ credential can delegate this credential to another user $\mathcal{P}_j$, giving $\mathcal{P}_j$ a Level-L credential. Delegator $\mathcal{P}_i$ can choose the attributes he adds in this delegation. Note that only the issuer $\mathcal{I}$ can issue a Level-1 credential, so we distinguish two cases: issuance (delegation of a Level-1 credential) and delegation of credential of level $L > 1$.

(2) $\mathcal{P}_i$ on input (DELEGATE, *sid*, *ssid*, $\vec{a}_1, \ldots, \vec{a}_L, \mathcal{P}_j$) with $\vec{a_L} \in \mathbb{A}_L^{n_L}$:
- If $L = 1$, $\mathcal{P}_i$ only proceeds if he is the issuer $\mathcal{I}$ with $sid = (\mathcal{I}, sid')$. If $L > 1$, $\mathcal{P}_i$ checks that he possesses a credential chain that signs $\vec{a}_1, \ldots, \vec{a}_{L-1}$. That is, he looks up a record $cred = (\langle \sigma_i, \vec{a}_i, cpk_i \rangle_{i=1}^{L-1}, csk_{L-1})$ in $\mathcal{L}_{\mathsf{cred}}$.
- Send $(sid, ssid, \vec{a}_1, \ldots, \vec{a_L})$ to $\mathcal{P}_j$ over $\mathcal{F}_{\mathsf{smt}}$.
- $\mathcal{P}_j$, upon receiving $(sid, ssid, \vec{a}_1, \ldots, \vec{a}_L)$ to $\mathcal{P}_j$ over $\mathcal{F}_{\mathsf{smt}}$ from $\mathcal{P}_i$, generate a fresh credential specific key pair $(cpk_L, csk_L) \leftarrow \mathsf{Sib}_L.\mathsf{Gen}(1^\kappa)$.
- Send cpk_L to $\mathcal{P}_i$ over $\mathcal{F}_{\mathsf{smt}}$.
- $\mathcal{P}_i$, upon receiving cpk_L from $\mathcal{P}_j$ over $\mathcal{F}_{\mathsf{smt}}$, computes $\sigma_L \leftarrow \mathsf{Sib}_{L-1}.\mathsf{Sign}_1(csk_{L-1}; cpk_L, \vec{a}_L)$ and sends $\langle \sigma_i, cpk_i \rangle_{i=1}^L$ to $\mathcal{P}_j$ over $\mathcal{F}_{\mathsf{smt}}$.
- $\mathcal{P}_j$, upon receiving $\langle \sigma_i, cpk_i \rangle_{i=1}^L$ from $\mathcal{P}_i$ over $\mathcal{F}_{\mathsf{smt}}$, verifies $\mathsf{Sib}_{i-1}.\mathsf{Verify}_1(cpk_{i-1}, \sigma_i, cpk_i, \vec{a}_i)$ for $i = 1, \ldots, L$. It stores $cred \leftarrow (\langle \sigma_i, \vec{a}_i, cpk_i \rangle_{i=1}^L, csk_L)$ in $\mathcal{L}_{\mathsf{cred}}$. Output (DELEGATE, *sid*, *ssid*, $\vec{a}_1, \ldots, \vec{a}_L, \mathcal{P}_i$).

Present. A user can present a credential she owns, while also signing a message m. The disclosed attributes are described by $\vec{a}_1, \ldots, \vec{a}_L$. Let $\vec{a}_i = a_{i,1}, \ldots, a_{i,n} \in (\mathbb{A} \cup \perp)^n$. If $a_{i,j} \in \mathbb{A}$, the user shows it possesses this attribute. If $a_{i,j} = \perp$, the user does not show the attribute. Let D be the set of indices of disclosed attributes, i.e., the set of pairs (i, j) where $a_{i,j} \neq \perp$.

(3) $\mathcal{P}_i$, upon receiving input (PRESENT, $sid, m, \vec{a}_1, \ldots, \vec{a}_L$) with $\vec{a}_i \in (\mathbb{A}_i \cup \perp)^{n_i}$ for $i = 1, \ldots, L$:
- Look up a credential $cred = (\langle \sigma_i, \vec{a}_i', cpk_i \rangle_{i=1}^L, csk_L)$ in $\mathcal{L}_{\mathsf{cred}}$, such that $\vec{a}_i \preceq \vec{a}_i{}'$ for $i = 1, \ldots, L$. Abort if no such credential was found.
- Create an attribute token by proving knowledge of the credential:

$$at \leftarrow \mathsf{NIZK}\Big\{(\sigma_1, \ldots, \sigma_L, cpk_1, \ldots, cpk_L, \langle a'_{i,j} \rangle_{i \notin D}, tag) : \bigwedge_{i=1}^{L} 1 = \mathsf{Sib}_{i-1}.\mathsf{Verify}_1(cpk_{i-1}, \sigma_i, cpk_i, a'_{i,1}, \ldots, a'_{i,n_i}) \wedge 1 = \mathsf{Sib}.\mathsf{Verify}_2(cpk_L, tag, m)\Big\}$$

- Output (TOKEN, *sid*, *at*).

Verify. A user can verify an attribute token by verifying the zero knowledge proof.

(4) $\mathcal{P}_i$, upon receiving input (VERIFY, *sid*, *at*, *m*, $\vec{a}_1, \ldots, \vec{a}_L$):
- Verify the zero-knowledge proof *at* with respect to m and $\vec{a}_1, \ldots, \vec{a}_L$. Set $f \leftarrow 1$ if valid and $f \leftarrow 0$ otherwise.
- Output (VERIFIED, *sid*, f).

4.3 Security of Π_{dac}

We now prove the security of our generic construction.

Theorem 4.1. *Our delegatable credentials protocol Π_{dac} securely realizes $\mathcal{F}_{\mathsf{dac}}$ (as defined in Section 3), in the $(\mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}_{\mathsf{crs}})$-hybrid model, provided that*

- Sib_i *is a secure sibling signature scheme (as defined in Section 2.5),*
- NIZK *is a simulation-sound zero-knowledge proof of knowledge.*

To prove Theorem 4.1, we have to show that there exists a simulator $\mathcal{S}$ as a function of $\mathcal{A}$ such that no environment can distinguish Π_{dac} and $\mathcal{A}$ from $\mathcal{F}_{\mathsf{dac}}$ and $\mathcal{S}$. The full proof is given in Appendix A, we present a proof sketch below.

Proof sketch. We sketch a satisfying simulator $\mathcal{S}$ and argue that with this simulator the real and ideal worlds are indistinguishable.

Setup. In the setup, $\mathcal{F}_{\mathsf{dac}}$ lets $\mathcal{S}$ define algorithms Present and Ver, and $\mathcal{F}_{\mathsf{dac}}$ will later use Present to generate attribute tokens. If the issuer is honest, $\mathcal{S}$ simiulates the issuer in the real world and knows its secret key *isk*. If $\mathcal{I}$ is corrupt, $\mathcal{S}$ can extract *isk* from π_{isk}, which is a part of the issuer public key, using the CRS trapdoor (decryption key). It defines Present to first issue a credential of the desired level and containing the requested attributes using *isk*, and to then prove knowledge of the credential as in the real world algorithm. It defines Ver as the real world verification algorithm.

Delegate. If both the delegator and delegatee are honest, $\mathcal{S}$ has to simulate the real world protocol without knowing the attribute values. In Π_{dac}, delegation takes place over secure channel $\mathcal{F}_{\mathsf{smt}}$. This allows $\mathcal{S}$ to simulate with dummy attribute values. If the delegator or delegatee is corrupt, $\mathcal{S}$ learns all attribute values and has all the information it needs to simulate the real world protocol.

Present. Credential presentation is non-interactive, meaning that there is no network communication to simulate for $\mathcal{S}$. We do have to argue that the output of the real world is indistinguishable from the output in the ideal world. $\mathcal{F}_{\text{dac}}$ first checks whether the user has the required credential for this presentation and aborts if this is not the case. In the real world, an honest signer also aborts if he does not possess the required credential for a presentation. Then, $\mathcal{F}_{\text{dac}}$ computes the attribute token using Present, and only outputs it if it verifies with Ver. This will always be the case for the Present and Ver that $\mathcal{S}$ defined: by completeness of the sibling signatures, it will create valid signatures, and by completeness of the NIZK, the resulting attribute token will be valid. This shows that the $\mathcal{F}_{\text{dac}}$ outputs an attribute token if and only if an honest signer would output an attribute token.

The attribute tokens that $\mathcal{F}_{\text{dac}}$ computed with Present differ from the ones computed in the real world: a real world party reuses one credential every time it signs, whereas Present creates a fresh credential for every presentation. By witness indistinguishability of NIZK, this is indistinguishable.

Verify. Verify again is non-interactive, so $\mathcal{S}$ does not have to simulate anything, but the outputs from the real and ideal world should be indistinguishable. $\mathcal{S}$ defined Ver as the real world verification algorithm, so both worlds use the same verification algorithm. There is one difference: $\mathcal{F}_{\text{dac}}$ prevents forgeries. This difference is indistinguishable under the unforgeability of the sibling signature schemes and the soundness of NIZK proof system. □

5 A CONCRETE INSTANTIATION USING PAIRINGS

We propose an efficient instantiation of our generic construction based on the Groth-Schnorr sibling signatures SibGS that we introduced in Sec. 2.5.

In the generic construction, we have a sibling signature scheme Sib_i for each delegation level i, where Sib_i must sign the public key of Sib_{i+1}. Groth signature scheme uses bilinear group $\Lambda = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2)$. Recall that Groth1 signs messages in $\mathbb{G}_1$ with a public key in $\mathbb{G}_2$, while Groth2 signs messages in $\mathbb{G}_2$ with a public key in $\mathbb{G}_1$. Therefore, we set Sib_{2n} to SibGS1 and Sib_{2n+1} to SibGS2. This means that we have attribute sets[1] $\mathbb{A}_{2n} = \mathbb{G}_1$ and $\mathbb{A}_{2n+1} = \mathbb{G}_2$.

In addition to the bilinear group, SibGS1 requires parameters $y_{1,1}, \ldots, y_{1,n+1} \in \mathbb{G}_1$, where n is the maximum number of attributes signed at an odd level ($n = max_{i=1,3,\ldots}(n_i)$), and SibGS2 requires $y_{2,1}, \ldots, y_{2,n+1} \in \mathbb{G}_2$, for n the maximum number of attributes signed at an even level ($n = max_{i=2,4,\ldots}(n_i)$). $\mathcal{F}_{\text{crs}}$ provides both the bilinear groups Λ and the $y_{i,j}$ values.

We consider Level-0 to be an even level and, therefore, the issuer key pair is $(ipk = g_2^{isk}, isk)$. The issuer must prove knowledge of its secret key isk in π_{isk} such that isk is online extractable. This extractability is required for the UC-security proof to work.

This can be achieved by verifiably encrypting the secret key to a public key encryption key in the CRS using the techniques of Camenisch and Shoup [11]. In the security proof, the simulator controls the CRS and hence knows the decryption key, and can therefore extract isk without rewinding.

If we only care about standalone security (rather than universal composability) and do not require the online extraction property, it is sufficient to prove $\pi_{isk} \leftarrow \text{SPK}\{isk : ipk = g_2^{isk}\}$ from which the secret key can be extracted in the proof using rewinding.

5.1 A Concrete Proof for the Attribute Tokens

What remains to show is how to efficiently instantiate the zero-knowledge proof that constitutes the attribute tokens. Since we instantiate Sib_{2i} with SibGS1 and Sib_{2i+1} with SibGS2, we can rewrite the proof we need to instantiate as follows.

$$
\begin{aligned}
at \leftarrow \text{NIZK}\Big\{(\sigma_1, \ldots, \sigma_L, cpk_1, \ldots, cpk_L, \langle a'_{i,j}\rangle_{i\notin D}, tag) : \\
\bigwedge_{i=1,3,\ldots}^{L} 1 = \text{SibGS1.Verify}_1(cpk_{i-1}, \sigma_i, cpk_i, a'_{i,1}, \ldots, a'_{i,n_i}) \\
\bigwedge_{i=2,4,\ldots}^{L} 1 = \text{SibGS2.Verify}_1(cpk_{i-1}, \sigma_i, cpk_i, a'_{i,1}, \ldots, a'_{i,n_i}) \\
\wedge\ 1 = \text{SibGSb.Verify}_2(cpk_L, tag, m)\Big\}
\end{aligned}
$$

The proof has three parts: First, it proves all the odd-level credential links by proving that σ_i is valid using SibGS1.Verify_1. Second, it proves the even-level credential links by proving that σ_i verifies with SibGS2.Verify_1. Finally, it proves that the user signed message m with SibGSb.Verify_2, where b depends on whether L is even or odd.

The abstract zero-knowledge proof can be efficiently instantiated with a generalized Schnorr zero-knowledge proof. Let $\sigma_i = (r_i, s_i, t_{i,1}, \ldots, t_{i,n_i+1})$. First, we use the fact that Groth is randomizable and randomize each signature to $(r'_i, s'_i, t'_{i,1}, \ldots, t'_{i,n_i+1})$. As r'_i is now uniform in the group, we can reveal the value rather than proving knowledge of it. Next, we use a Schnorr-type proof depicted in Fig. 3 to prove knowledge of the s and t values of the signatures, the undisclosed attributes, the credential public keys, and the credential secret key. The concrete zero-knowledge proof contains the same parts as described for the abstract zero-knowledge proof. The third part, proving knowledge of tag, is somewhat hidden. Recall that we instantiate SibGSb.Verify_2 with Schnorr signatures, which means the signature is a proof of knowledge of csk_L. This can efficiently be combined with other two parts of the proof: instead of proving knowledge of cpk_L, we prove knowledge of csk_L.

5.2 Optimizing Attribute Token Computation

There is a lot of room for optimization when computing zero-knowledge proofs such as the one depicted in Fig. 3. We describe how to efficiently compute this specific proof, but many of these optimizations will be applicable to other zero-knowledge proofs in pairing-based settings.

Computing attribute tokens. The pairing operation is the most expensive operation in bilinear groups, so for the efficiency of the scheme it is beneficial to minimize the amount of pairings computed. We can use some optimizations in computing the zero-knowledge

[1] Alternatively, one could define a single attribute set $\mathbb{A}$ for all levels and use injective functions $f_1 : \mathbb{A} \to \mathbb{G}_1$ and $f_2 : \mathbb{A} \to \mathbb{G}_2$, such as setting $\mathbb{A} = \mathbb{Z}_q$ and $f_1(a) = g_1^a$, $f_2(a) = g_2^a$), but for ease of presentation we omit this step and work directly with attributes in $\mathbb{G}_1$ and $\mathbb{G}_2$.

$$\mathsf{SPK}\Big\{(\langle s'_i, t'_{i,j}\rangle_{i=1,\ldots,L,j=1,\ldots,n_i}, \langle a_{i,j}\rangle_{i\notin D}, \langle cpk_i\rangle_{i=1,\ldots,L-1}, csk_L):$$

$$\bigwedge_{i=1,3,\ldots}^{L}\Bigg(e(y_{1,1},g_2)\big[e(g_1,ipk)\big]_{i=1} = e(\underline{s'_i}, r'_i)\big[e(g_1^{-1},\underline{cpk_{i-1}})\big]_{i\neq 1} \wedge$$

$$1_{\mathbb{G}_t}\big[e(y_{1,1},ipk)\big]_{i=1} = e(\underline{t'_{i,1}}, r'_i)\big[e(\underline{cpk_i}, g_2^{-1})\big]_{i\neq L}\big[e(g_1, g_2^{-1})^{\underline{csk_i}}\big]_{i=L}\big[e(y_{1,1}^{-1},\underline{cpk_{i-1}})\big]_{i\neq 1} \wedge$$

$$\bigwedge_{j:(i,j)\in D} e(a_{i,j},g_2)\big[e(y_{1,j+1},ipk)\big]_{i=1} = e(\underline{t'_{i,j+1}}, r'_i)\big[e(y_{1,j+1}^{-1},\underline{cpk_{i-1}})\big]_{i\neq 1} \wedge$$

$$\bigwedge_{j:(i,j)\notin D} 1_{\mathbb{G}_t}\big[e(y_{1,j+1},ipk)\big]_{i=1} = e(\underline{t'_{i,j+1}}, r'_i)e(\underline{a_{i,j}}, g_2^{-1})\big[e(y_{1,j+1}^{-1},\underline{cpk_{i-1}})\big]_{i\neq 1}\Bigg) \wedge$$

$$\bigwedge_{i=2,4,\ldots}^{L}\Bigg(e(g_1,y_{2,1}) = e(r'_i,\underline{s'_i})e(\underline{cpk_{i-1}}, g_2^{-1}) \wedge 1_{\mathbb{G}_t} = e(r'_i,\underline{t'_{i,1}})e(\underline{cpk_{i-1}}, y_{2,1}^{-1})\big[e(g_1^{-1},\underline{cpk_i})\big]_{i\neq L}\big[e(g_1^{-1},g_2)^{\underline{csk_i}}\big]_{i=L} \wedge$$

$$\bigwedge_{j:(i,j)\in D} e(g_1,a_{i,j}) = e(r'_i,\underline{t'_{i,j+1}})e(\underline{cpk_{i-1}}, y_{2,j+1}^{-1}) \wedge \bigwedge_{j:(i,j)\notin D} 1_{\mathbb{G}_t} = e(r'_i,\underline{t'_{i,j+1}})e(\underline{cpk_{i-1}}, y_{2,j+1}^{-1})e(g_1^{-1},\underline{a_{i,j}})\Bigg)\Big\}(sp, r'_1,\ldots,r'_L, m).$$

Figure 3: Efficient instantiation of the NIZK used to generate attribute tokens (witness underlined for clarity).

proof that remove the need to compute any pairings. As a small example, suppose we prove $\mathsf{SPK}\{x : z = e(x,b)\}$. The standard way to compute this is taking $r_x \xleftarrow{\$} \mathbb{G}_1$, computing $\mathsf{com} \leftarrow e(r_x, b)$, $c \leftarrow \mathsf{H}(\mathsf{com},\ldots)$, and $\mathsf{res}_x \leftarrow r_x \cdot x^c$. We can compute the same values without computing the pairing by precomputing $e(g,b)$, taking $\rho \xleftarrow{\$} \mathbb{Z}_q$ and setting $\mathsf{com} \leftarrow e(g_1,b)^\rho$ and $\mathsf{res} \leftarrow g_1^\rho x^c$.

To prove knowledge of a Groth signature, we must prove $z = e(x,r')$, where r' is the randomized r-value of the Groth signature. If we try to apply the previous trick, we set $\rho_x \xleftarrow{\$} \mathbb{Z}_q$, $\mathsf{com}_x \leftarrow e(g_1,r')^{\rho_x}$. However, now we cannot precompute $e(g_1,r')$ since r' is randomized before every proof. We can solve this by remembering the randomness used to randomize the Groth signature. Let $r' = r^{\rho_\sigma}$, we can compute $\mathsf{com}_x \leftarrow e(g_1,r)^{\rho_\sigma\cdot\rho_x}$ by precomputing $e(g_1,r)$. The full pseudocode for computing the proofs using these optimizations is given in Fig. 4.

Verifying attribute tokens. In verification, computing pairings is unavoidable, but there are still tricks to keep verification efficient. The pairing function is typically instantiated with the tate pairing, which consists of two parts: Miller's algorithm $\hat{t}(\cdot)$ and the final exponentiation $\mathsf{fexp}(\cdot)$ [23]. Both parts account for roughly half the time required to compute a pairing[2]. When computing the product of multiple pairings, we can compute the Miller loop for every pairing and then compute the final exponentiation only once for the whole product. This means that computing the product of three pairings is roughly equally expensive as computing two individual pairings.

Fig. 5 shows how to verify attribute tokens efficiently using this observation. When we write $e(a,b)$ in the pseudocode, it means we can precompute the value.

[2]We verified this by running `bench_pair.c` of the AMCL library (github.com/miracl/amcl) using the BN254 curve.

5.3 Efficiency Analysis of Our Instantiation

We now analyze the efficiency of our construction. Namely, we calculate the number of pairing operations and (multi-) exponentiations in different groups that is required to compute and verify attribute tokens. We also compute the size of credentials and attribute tokens with respect to a delegation level and number of attributes. We provide concrete timings for our prototype implementation in C that generates and verifies Level-2 attribute tokens in Section 6.3.

Let d_i and u_i denote the amount of disclosed and undisclosed attributes at delegation level i, respectively, and we define $n_i = d_i + u_i$.

Computational efficiency. Let us count the operations required to compute and verify attribute tokens. For operations we use the following notation. We use $X\{\mathbb{G}_1^j\}$, $X\{\mathbb{G}_2^j\}$, and $X\{\mathbb{G}_t^j\}$ to denote X j-multi-exponentiations in the respective group; $j = 1$ means a simple exponentiation. We denote as E^k a k-pairing product that we can compute with k-Miller loops and a single shared final exponentiation.

Setup. During the setup, the issuer chooses its root issuer key isk and computes $ipk \leftarrow g_2^{isk}$, costing $1\{\mathbb{G}_2\}$.

Delegation. Delegation of a credential includes generating a key and a signature on the public key and a set of attributes:

- for even i the cost is $1\{\mathbb{G}_1\} + (n_i+2)\{\mathbb{G}_2\} + (n_i+1)\{\mathbb{G}_2^2\}$,
- for odd i the cost is $1\{\mathbb{G}_2\} + (n_i+2)\{\mathbb{G}_1\} + (n_i+1)\{\mathbb{G}_1^2\}$.

Signature verification for Level-i costs $n_i \cdot E^3$ plus E^2 or E^3, depending on if the pairing with the public key was pre-computed or not.

Computing attribute tokens (Presentation). Randomizing σ_i costs $(n_i+2)\cdot\{\mathbb{G}_1\} + 1\{\mathbb{G}_2\}$ for odd i and $1\{\mathbb{G}_1\} + (n_i+2)\{\mathbb{G}_2\}$ for even i. Computing the com-values for Level-1 costs $(1+d_i)\{\mathbb{G}_t\} +$

input: $\langle r_i, s_i, \langle t_{i,j}\rangle_{j=1}^{n_1+1}\rangle_{i=1}^{L}, csk_L, \langle cpk_i\rangle_{i=1}^{L}, \langle a_{i,j}\rangle_{i=1,\ldots,L,j=1,\ldots,n_i}, D, sp, m$
for $i = 1, \ldots, L$ **do** ▷ Randomize σ_i
  $\rho_{\sigma_i} \xleftarrow{\$} \mathbb{Z}_q, r_1' \leftarrow r_i^{\rho_{\sigma_i}}, s_i' \leftarrow s_i^{\frac{1}{\rho_{\sigma_i}}}$
  for $j = 1, \ldots n_i + 1$ **do**
    $t_{i,j}' \leftarrow t_{i,j}^{\frac{1}{\rho_{\sigma_i}}}$
  end for
end for
$\langle \rho_{s_i}, \langle \rho_{t_{i,j}}\rangle_{j=1}^{n_i+1}, \langle \rho_{a_{i,j}}\rangle_{j=1}^{n_i}\rangle_{i=1}^{L}, \langle \rho_{cpk_i}\rangle_{i=1}^{L-1}, \rho_{csk_L} \xleftarrow{\$} \mathbb{Z}_q$
for $i = 1, 3, \ldots, L$ **do** ▷ Compute com-values for odd-level σ_i
  $\mathsf{com}_{i,1} \leftarrow e(g_1, r_i)^{\rho_{\sigma_i}\cdot\rho_{s_i}} \left[\cdot\, e(g_1^{-1}, g_2)^{\rho_{cpk_{i-1}}}\right]_{i\neq 1}$
  $\mathsf{com}_{i,2} \leftarrow e(g_1, r_i)^{\rho_{\sigma_i}\cdot\rho_{t_{i,1}}} \cdot e(g_1, g_2^{-1})^{\rho_{cpk_i}} \left[\cdot\, e(y_{1,1}, g_2)^{\rho_{cpk_{i-1}}}\right]_{i\neq 1}$
  for $j = 1, \ldots, n_i$ **do**
    if $(i,j) \in D$ **then** ▷ Attribute $a_{i,j}$ is disclosed
      $\mathsf{com}_{i,j+2} \leftarrow e(g_1, r_i)^{\rho_{\sigma_i}\cdot\rho_{t_{i,j+1}}} \left[\cdot\, e(y_{1,j+1}, g_2)^{\rho_{cpk_{i-1}}}\right]_{i\neq 1}$
    else ▷ Attribute $a_{i,j}$ is hidden
      $\mathsf{com}_{i,j+2} \leftarrow e(g_1, r_i)^{\rho_{\sigma_i}\cdot\rho_{t_{i,j+1}}} \cdot e(g_1, g_2^{-1})^{\rho_{a_{i,j}}} \left[\cdot\, e(y_{1,j+1}, g_2)^{\rho_{cpk_{i-1}}}\right]_{i\neq 1}$
    end if
  end for
end for
for $i = 2, 4, \ldots, L$ **do** ▷ Compute com-values for even-level σ_i
  $\mathsf{com}_{i,1} \leftarrow e(r_i, g_2)^{\rho_{\sigma_i}\cdot\rho_{s_i}} e(g_1, g_2^{-1})^{\rho_{cpk_{i-1}}}$
  $\mathsf{com}_{i,2} \leftarrow e(r_i, g_2)^{\rho_{\sigma_i}\cdot\rho_{t_{i,1}}} e(g_1, y_{2,1}^{-1})^{\rho_{cpk_{i-1}}} e(g_1^{-1}, g_2)^{\rho_{cpk_i}}$
  for $j = 1, \ldots, n_i$ **do**
    if $(i,j) \in D$ **then** ▷ Attribute $a_{i,j}$ is disclosed
      $\mathsf{com}_{i,j+2} \leftarrow e(g_1, y_{2,j+1}^{-1})^{\rho_{cpk_{i-1}}} \cdot e(r_i, g_2)^{\rho_{\sigma_i}\cdot\rho_{t_{i,j+1}}}$
    else ▷ Attribute $a_{i,j}$ is hidden
      $\mathsf{com}_{i,j+2} \leftarrow e(g_1, y_{2,j+1}^{-1})^{\rho_{cpk_{i-1}}} \cdot e(r_i, g_2)^{\rho_{\sigma_i}\cdot\rho_{t_{i,j+1}}} \cdot e(g_1^{-1}, g_2)^{\rho_{a_{i,j}}}$
    end if
  end for
end for
$c \leftarrow \mathsf{H}(sp, ipk, \langle r_i', \langle \mathsf{com}_{i,j}\rangle_{j=1}^{n_i+2}\rangle_{i=1}^{L}, \langle a_{i,j}\rangle_{(i,j)\in D}, m)$ ▷ Fiat-Shamir hash
for $i = 1, 3, \ldots, L$ **do** ▷ Compute res-values for odd-level σ_i
  $\mathsf{res}_{s_i} = g_1^{\rho_{s_i}} s_i^c, \quad \left[\mathsf{res}_{cpk_i} = g_1^{\rho_{cpk_i}} cpk_i^c\right]_{i\neq L}, \left[\mathsf{res}_{csk_i} = \rho_{cpk_i} + c \cdot csk_i\right]_{i=L}$
  for $j = 1, \ldots, n_i + 1$ **do**
    $\mathsf{res}_{t_{i,j}} = g_1^{\rho_{t_{i,j}}} t_{i,j}^c$
  end for
  for $j = 1, \ldots, n_i$ with $(i,j) \notin D$ **do**
    $\mathsf{res}_{a_{i,j}} = g_1^{\rho_{a_{i,j}}} a_{i,j}^c$
  end for
end for
for $i = 2, 4, \ldots, L$ **do** ▷ Compute res-values for even-level σ_i
  $\mathsf{res}_{s_i} = g_2^{\rho_{s_i}} s_i^c, \quad \left[\mathsf{res}_{cpk_i} = g_2^{\rho_{cpk_i}} cpk_i^c\right]_{i\neq L}, \left[\mathsf{res}_{csk_i} = \rho_{cpk_i} + c \cdot csk_i\right]_{i=L}$
  for $j = 1, \ldots, n_i + 1$ **do**
    $\mathsf{res}_{t_{i,j}} = g_2^{\rho_{t_{i,j}}} t_{i,j}^c$
  end for
  for $j = 1, \ldots, n_i$ with $(i,j) \notin D$ **do**
    $\mathsf{res}_{a_{i,j}} = g_2^{\rho_{a_{i,j}}} a_{i,j}^c$
  end for
end for
output: $c, \langle r_i', \mathsf{res}_{s_i}, \langle \mathsf{res}_{t_{i,j}}\rangle_{j=1}^{n_i+1}\rangle_{i=1}^{L}, \langle \mathsf{res}_{a_{i,j}}\rangle_{(i,j)\notin D}, \langle \mathsf{res}_{cpk_i}\rangle_{i=1}^{L-1}, \mathsf{res}_{csk_L}$

Figure 4: Pseudocode for efficiently computing attribute tokens.

input: $c, \langle r'_i, \mathsf{res}_{s_i}, \langle \mathsf{res}_{t_{i,j}} \rangle_{j=1}^{n_i+1} \rangle_{i=1}^{L}, \langle \mathsf{res}_{a_{i,j}} \rangle_{(i,j)\notin D}, \langle \mathsf{res}_{cpk_i} \rangle_{i=1}^{L-1}, \mathsf{res}_{csk_L},$
$\langle a_{i,j} \rangle_{(i,j\in D)}, D, sp, m$
for $i = 1, 3, \ldots, L$ **do** ▷ Recompute com-values for odd-level σ_i
  $\mathsf{com}_{i,1} \leftarrow \mathsf{fexp}(\hat{t}(\mathsf{res}_{s_i}, r'_i)\big[\cdot \hat{t}(g_1^{-1}, \mathsf{res}_{cpk_{i-1}})\big]_{i\neq 1}) \cdot (e(y_{1,1}, g_2)\big[\cdot e(g_1, ipk)\big]_{i=1})^{-c}$
  $\mathsf{com}_{i,2} \leftarrow \mathsf{fexp}(\hat{t}(\mathsf{res}_{t_{i,1}}, r'_i)\big[\cdot \hat{t}(y_{1,1}, \mathsf{res}_{cpk_{i-1}})\big]_{i\neq 1}\big[\cdot \hat{t}(\mathsf{res}_{cpk_i}, g_2^{-1})\big]_{i\neq L}\big[\cdot e(g_1, g_2^{-1})^{\mathsf{res}_{csk_i}}\big]_{i=L}\big[\cdot e(y_{1,1}, ipk)^{-c}\big]_{i=1}$
  for $j = 1, \ldots, n_i$ **do**
    if $(i,j) \in D$ **then** ▷ Attribute $a_{i,j}$ is disclosed
      $\mathsf{com}_{i,j+2} \leftarrow \mathsf{fexp}(\hat{t}(\mathsf{res}_{t_{i,j+1}}, r'_i)\big[\cdot \hat{t}(y_{1,j+1}, \mathsf{res}_{cpk_{i-1}})\big]_{i\neq 1}) \cdot (e(a_{i,j}, g_2)\big[e(y_{1,j+1}, ipk)\big]_{i=1})^{-c}$
    else ▷ Attribute $a_{i,j}$ is hidden
      $\mathsf{com}_{i,j+2} \leftarrow \mathsf{fexp}(\hat{t}(\mathsf{res}_{t_{i,j+1}}, r'_i) \cdot \hat{t}(\mathsf{res}_{a_{i,j}}, g_2^{-1})\big[\cdot \hat{t}(y_{1,j+1}, \mathsf{res}_{cpk_{i-1}})\big]_{i\neq 1})\big[\cdot e(y_{1,j+1}, ipk)^{-c}\big]_{i=1}$
    end if
  end for
end for
for $i = 2, 4, \ldots, L$ **do** ▷ Compute com-values for even-level σ_i
  $\mathsf{com}_{i,1} \leftarrow \mathsf{fexp}(\hat{t}(r'_i, \mathsf{res}_{s_i}) \cdot \hat{t}(\mathsf{res}_{cpk_{i-1}}, g_2^{-1})) \cdot e(g_1, y_{2,1})^{-c}$
  $\mathsf{com}_{i,2} \leftarrow \mathsf{fexp}(\hat{t}(r'_i, \mathsf{res}_{t_{i,1}}) \cdot \hat{t}(\mathsf{res}_{cpk_{i-1}}, y_{2,1}^{-1})\big[\cdot \hat{t}(g_1^{-1}, \mathsf{res}_{cpk_i})\big]_{i\neq L})\big[\cdot e(g_1^{-1}, g_2)^{\mathsf{res}_{csk_i}}\big]_{i=L}$
  for $j = 1, \ldots, n_i$ **do**
    if $(i,j) \in D$ **then** ▷ Attribute $a_{i,j}$ is disclosed
      $\mathsf{com}_{i,j+2} \leftarrow \mathsf{fexp}(\hat{t}(\mathsf{res}_{cpk_{i-1}}, y_{2,j+1}^{-1}) \cdot \hat{t}(r'_i, \mathsf{res}_{t_{i,j+1}})) \cdot e(g_1, a_{i,j})^{-c}$
    else ▷ Attribute $a_{i,j}$ is hidden
      $\mathsf{com}_{i,j+2} \leftarrow \mathsf{fexp}(\hat{t}(\mathsf{res}_{cpk_{i-1}}, y_{2,j+1}^{-1}) \cdot \hat{t}(r'_i, \mathsf{res}_{t_{i,j+1}}) \cdot \hat{t}(g_1^{-1}, \mathsf{res}_{a_{i,j}}))$
    end if
  end for
end for
$c' \leftarrow \mathsf{H}(sp, ipk, \langle r'_i, \langle \mathsf{com}_{i,j} \rangle_{j=1}^{n_i+2} \rangle_{i=1}^{L}, \langle a_{i,j} \rangle_{(i,j)\in D}, m)$ ▷ Fiat-Shamir hash
output: $c = c'$

Figure 5: Pseudocode for efficiently verifying attribute tokens.

$(1+u_i)\{\mathbb{G}_t^2\}$. The com-values for Level-i for $i > 1$ cost $(1+d_i)\{\mathbb{G}_t^2\} + (1+u_i)\{\mathbb{G}_t^3\}$. Computing the res-values for odd i costs $(2+n_i)\{\mathbb{G}_1^2\}$, and for even i it costs $(2+n_i)\{\mathbb{G}_2^2\}$, except the last level, where $1\{\mathbb{G}_1^2\}$ or $1\{\mathbb{G}_2^2\}$ can be saved when L is even or odd, respectively.

If we consider a practical example, where we show Level-2 credentials with attributes only on Level-1 (meaning that $n_2 = 0$), computing the attribute token costs very roughly $3n_1 + 13$ exponentiations, and more precisely: $(3+n_1)\{\mathbb{G}_1\} + (2+n_1)\{\mathbb{G}_1^2\} + 3\{\mathbb{G}_2\} + 1\{\mathbb{G}_2^2\} + (1+d_1)\{\mathbb{G}_t\} + (2+u_1)\{\mathbb{G}_t^2\} + 1\{\mathbb{G}_t^3\}$.

Verifying attribute tokens. Verifying the first credential link costs $(1+d_1)E + (1+u_1)E^2 + (2+n_1)\{\mathbb{G}_t\}$ and one final exponentiation. Every next level adds $(1+d_i)E^2 + (1+u_i)E^3 + (1+d_i)\{\mathbb{G}_t\}$, except the last level, which costs $(2+d_i)E^2 + u_iE^3 + (2+d_i)\{\mathbb{G}_t\}$.

For the same practical example with two levels, to verify a Level-2 attribute token will cost very roughly $n_1 + 4$ pairings and $n_1 + 4$ exponentiations, and more precisely: $(1+d_1)E + (3+u_1)E^2 + (4+n_1)\{\mathbb{G}_t\}$.

We summarize the above efficiency analysis in Table 1. We also estimate timings from running some benschmarks for the C version of Apache Milagro Cryptographic Library (AMCL)[3] with a 254-bit Barreto-Naehrig curve [2] on a 3.1GHz Intel I7-5557U laptop CPU. We compared our timings calculated according to Table 1 with the real implementation for two levels with different amount of attributes (see Table 2 for comparison). The figures from Table 2 show that our estimates are quite accurate and even a bit conservative.

Size of attribute tokens To count the size of an attribute token we use the following notation. We use $X[\mathbb{G}_1]$ and $X[\mathbb{G}_2]$ to denote X group elements from the respective group. The attribute token proves knowledge of every credential link, so the token grows in the credential level.

First, we look at credential links without attributes. For every level a credential link adds 4 group elements: $3[\mathbb{G}_1] + 1[\mathbb{G}_2]$ for an odd and $1[\mathbb{G}_1] + 3[\mathbb{G}_2]$ for an even level, respectively. Additionally, a token has 2 elements from $\mathbb{Z}_q$. This means that for even L, an attribute token generated from a Level-L credential without attributes takes $(2L)[\mathbb{G}_1] + (2L-1)[\mathbb{G}_2] + 2\mathbb{Z}_q$.

Every attribute added to an odd level credential link adds one group element, if it is disclosed, and two elements, if this attribute remains hidden. For the odd levels these are the elements from $[\mathbb{G}_1]$ and for even levels - from $[\mathbb{G}_2]$. This means that for even L, an attribute token generated from a Level-L credential takes $(2L + \sum_{i=1,3,\ldots}^{L-1}(n_i+u_i))[\mathbb{G}_1] + (2L-1+\sum_{i=2,4,\ldots}^{L}(n_i+u_i))[\mathbb{G}_2] + 2\mathbb{Z}_p$.

[3] github.com/miracl/amcl

Algorithm	Operations	Total time estimate (ms)
SETUP	$1\{\mathbb{G}_2\}$	1.21
DELEGATE	For each odd Level-i: $1\{\mathbb{G}_2\} + (n_i + 2)\{\mathbb{G}_1\} + (n_i + 1)\{\mathbb{G}_1^2\}$	$2.96 + 1.21n_i$
	For each even Level-i: $1\{\mathbb{G}_1\} + (n_i + 2)\{\mathbb{G}_2\} + (n_i + 1)\{\mathbb{G}_2^2\}$	$5.27 + 3.52n_i$
PRESENT	$\sum_{i=1,3,..}^{L} \left(1\{\mathbb{G}_2\} + (n_i + 2)\{\mathbb{G}_1\} + (1 + d_i)\{\mathbb{G}_t^2\} + (1 + u_i)\{\mathbb{G}_t^3\} + (2 + n_i)\{\mathbb{G}_1^2\}\right)$	$\sum_{i=1,3,..}^{L} \left(13.63 + 3.89d_i + 6.11u_i + 1.21n_i\right) +$
	$\sum_{i=2,4,..}^{L} \left(1\{\mathbb{G}_1\} + (n_i + 2)\{\mathbb{G}_2\} + (1 + d_i)\{\mathbb{G}_t^2\} + (1 + u_i)\{\mathbb{G}_t^3\} + (2 + n_i)\{\mathbb{G}_2^2\}\right)$	$\sum_{i=2,4,..}^{L} \left(17.58 + 3.89d_i + 6.11u_i + 3.52n_i\right)$
VERIFY	$(1 + d_1)E + (3 + u_1 + d_L)E^2 + u_L E^3 + (4 + n_1 + d_L)\{\mathbb{G}_t\} + \sum_{i=2,3,..}^{(L-1)} \left((1 + d_i)E^2 + (1 + u_i)E^3 + (1 + d_i)\{\mathbb{G}_t\}\right)$	$21.65 + 2.36d_1 + 3.91u_1 + 1.89n_1 + 5.80d_L + 5.48u_L + \sum_{i=2,3,..}^{(L-1)} \left(11.28 + 5.80d_i + 5.48u_i\right)$

Table 1: Performance evaluation and timing estimations, where d_i and u_i denote the amount of disclosed and undisclosed attributes at delegation level i, respectively, and $n_i = d_i + u_i$; $X\{\mathbb{G}_1^j\}$, $X\{\mathbb{G}_2^j\}$, and $X\{\mathbb{G}_t^j\}$ denote X j-multi-exponentiations in the respective group; $j = 1$ means a simple exponentiation. E^k denote a k-pairing product that we can compute with k-Miller loops and a single shared final exponentiation; $k = 1$ means a single pairing. Benchmarks are (all in ms): $1\{\mathbb{G}_1\} = 0.54; 1\{\mathbb{G}_1^2\} = 0.67; 1\{\mathbb{G}_2\} = 1.21; 1\{\mathbb{G}_2^2\} = 2.31; 1\{\mathbb{G}_t\} = 1.89; 1\{\mathbb{G}_t^2\} = 3.89; 1\{\mathbb{G}_t^3\} = 6.11; 1E = 2.36; 1E^2 = 3.91; 1E^3 = 5.48$.

6 APPLICATION TO PERMISSIONED BLOCKCHAINS

In this section, we describe a practical application of delegatable credentials with attributes to a membership service for a permissioned blockchain. We also report on the implementation of this scheme, demonstrating the practicality of our construction for real-world applications.

Blockchain is a distributed immutable ledger widely used in cryptocurrencies and beyond for different kinds of transactions. Blockchain is the basis for Bitcoin [32], which greatly helped to popularize distributed cryptographic protocols. Bitcoin is an example of a permissionless blockchain, i.e., anyone can submit transactions and anyone with the sufficient computational power can join in maintaining the ledger. However, for some applications including, in particular, many enterprise scenarios, only designated parties should be allowed to submit transactions or be able to modify the state of the blockchain. Thus, mechanisms for identity verification and for moderation of who can add and modify the blockchain entries need to be in place. Furthermore, it is often necessary that all transactions can be audited.

These requirements are addressed by permissioned blockchains, sometimes also called *private* blockchains. To this end, a permissioned blockchain entails a so-called *membership service* that issues credentials to the members of the chain and provides mechanisms to enable transaction signing, authentication, access control, revocation of credentials, and auditing of the transactions. However, all transactions being traceable can violate the privacy and security requirements. Therefore, anonymous credentials are a perfect fit to implement a privacy-preserving membership service. Below we describe how to implement a membership service with ordinary anonymous credentials and then show how to extend it to incorporate delegateable credentials.

6.1 Privacy-Preserving Membership Service

The membership service realized with an anonymous credential scheme works as follows.

Setup. The Certificate Authority (CA) is set up by generating the signing key pair and making the public key available to the blockchain participants.

Certificate Issuance. A blockchain participant generates a secret key and creates a request for a membership certificate. The CA issues a membership certificate as an anonymous credential. The certificate also contains the attributes associated with the participant. These can then be used to implement (attribute-based) access control for transactions. The certificate is stored together with the corresponding credential secret key by the participant.

Signing Transactions. When a blockchain member needs to sign a transaction, it generates a fresh unlinkable presentation token that: 1) signs the transaction content; 2) proves a possession of a valid membership credential issued by the CA; 3) discloses the attributes that are required by the access control policy for the transaction.

To enable certificate revocation and auditing, the token can also prove in zero-knowledge: 4) that the certificate was not revoked with respect to the revocation information published by the membership service (or a designated revocation authority); 5) provide a ciphertext that contains the credential identifier encrypted under the auditor's public encryption key (so that only the auditor can decrypt it) and a ZK proof that the same credential identifier is contained in the membership certificate, without disclosing the identifier itself.

Implementing a membership service like this preserves transaction privacy and unlinkability and enables transaction auditing and membership revocation. However, when many different organizations run blockchain and many users from these organizations participate in transactions, it is hardly practical to deploy a single

membership service (the CA) because of the issuance workload and also because it introduces a single point of trust and failure. While introducing a two or more level hierarchy of CAs could be a solution, this approach would have a serious impact on the privacy and confidentiality of the system, as argued earlier. In the next section we, therefore, describe how to use delegatable credentials for this task to preserve privacy and unlinkability of transactions.

6.2 Hierarchical Membership Service from Delegatable Credentials with Attributes

A membership service implemented with a two-level DAC is as follows. The root CA will be the issuer providing Level-1 credentials with the suitably chosen attributes to the local CAs. The local CAs issue Level-2 credentials to the blockchain members, certifying the attributes of every member. Blockchain members can use Level-2 credentials to unlinkably sign transactions, selectively disclosing Level-2 attributes (and possibly Level-1 attributes, if required by the application). Signatures on transactions can thus be verified with the root CA's public key only, without leaking any information about the local CAs. With this approach, the issuance and identity management workload is distributed among the different organization running the system without compromising privacy. The number of delegation levels can be increased to support different hierarchies based on the organizational structure and will still preserve the privacy of all intermediate CAs.

Enhancing Trust in the Root CA by Distribution. To avoid a single point of trust and failure at the root CA, the first level issuance procedures (issuance of Level-1 credentials) can be realized as a multiparty computation. Due the algebraic properties of the Groth signature scheme [29] that we use, such multiparty computation can be efficiently implemented using known techniques [5, 22]. Essentially, the parties will have to generate a random share, compute its inverse and then three distributed exponentiations. Also, issuing root credentials is probably the least frequent operation and less critical for the system's performance compared to the delegations at other levels. Thus the loss of efficiency due to the distribution of the root issuance will hardly have an effect in real deployments.

6.3 Implementation and Performance Analysis

We have implemented a prototype of our concrete instantiation for delegatable credentials in the C programming language, using the Apache Milagro Cryptographic Library (AMCL) with a 254-bit Barreto-Naehrig curve [2]. This prototype generates and verifies Level-2 attribute tokens. The prototype shows the practicality of our construction: generating an attribute token without attributes takes only 27 ms, and verification requires only 20 ms, on a 3.1GHz Intel I7-5557U laptop CPU. Table 2 shows performance figures when presenting tokens with attributes. Adding undisclosed attributes in the first credential link (that is, increasing n_1) adds roughly 6 ms to the token generation time per attribute, while adding undisclosed attributes in the second link (thus increasing n_2) adds 11 ms. For verification, every added undisclosed attribute increases verification time by 5 ms. Table 2 also shows that our estimated timings in Table 1 are accurate: the estimated values are close to the measured timings and our estimates are even a bit conservative. We plan to release our prototype implementation as open source software. Currently it is available upon request.

n_1	n_2	PRESENT	VERIFY	EST. PRES.	EST. VERIFY
0	0	26.9 ms	20.2 ms	31.21 ms	21.65 ms
1	0	32.7 ms	25.4 ms	38.53 ms	27.45 ms
2	0	38.1 ms	30.9 ms	45.85 ms	33.25 ms
3	0	44.0 ms	36.1 ms	53.17 ms	39.05 ms
4	0	49.5 ms	41.4 ms	60.49 ms	44.85 ms
0	1	38.6 ms	24.8 ms	40.84 ms	27.13 ms
0	2	49.4 ms	29.2 ms	50.47 ms	32.61 ms
0	3	61.5 ms	34.1 ms	60.10 ms	38.09 ms
0	4	72.6 ms	38.7 ms	69.73 ms	43.57 ms
1	1	43.7 ms	30.1 ms	48.16 ms	32.93 ms
2	1	49.3 ms	35.4 ms	55.48 ms	38.73 ms

Table 2: Performance measurements of presenting and verifying Level-2 credentials, and our estimated timings following the computation of Table 1. No attributes are disclosed.

7 CONCLUSION

The first practical delegatable credential system with attributes presented in this paper addresses the basic privacy and security needs of a public key infrastructure and, in particular, the requirements of a membership service of a permissioned blockchain. However, there are a number of additional functionalities that could be considered, such as key life cycle management, revocation, and support for auditable tokens. We expect that the solutions for these extensions known for the ordinary anonymous credentials to be applicable here as well. Of course, any of these extensions would require to modify our ideal functionality $\mathcal{F}_{\text{dac}}$, as would the extension of a distributed issuance of Level-1 credentials. One way to do it is to extend our ideal functionality $\mathcal{F}_{\text{dac}}$ to accept as input and also output commitments to attribute values, following the recent work by Camenisch et al. [7]. This would allow for a modular construction of a delegatable credential scheme with the extensions just discussed. We consider all of this future work.

8 ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers for their helpful comments. This work was supported by the European Commission through the Seventh Framework Programme, under grant agreements #321310 for the ERC grant PERCY.

REFERENCES

[1] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. 2010. Structure-Preserving Signatures and Commitments to Group Elements. In *CRYPTO 2010 (LNCS)*, Tal Rabin (Ed.), Vol. 6223. Springer, Heidelberg, 209–236.

[2] Paulo S. L. M. Barreto and Michael Naehrig. 2006. Pairing-Friendly Elliptic Curves of Prime Order. In *SAC 2005 (LNCS)*, Bart Preneel and Stafford Tavares (Eds.), Vol. 3897. Springer, Heidelberg, 319–331.

[3] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. 2009. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO 2009 (LNCS)*, Shai Halevi (Ed.), Vol. 5677. Springer, Heidelberg, 108–125.

[4] Mihir Bellare and Gregory Neven. 2006. Multi-signatures in the plain public-Key model and a general forking lemma. In *ACM CCS 06*, Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati (Eds.). ACM Press, 390–399.

[5] Assaf Ben-David, Noam Nisan, and Benny Pinkas. 2008. FairplayMP: a system for secure multi-party computation. In *ACM CCS 08*, Peng Ning, Paul F. Syverson, and Somesh Jha (Eds.). ACM Press, 257–266.
[6] Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, Anja Lehmann, Gregory Neven, Christian Paquin, and Franz-Stefan Preiss. 2014. Concepts and languages for privacy-preserving attribute-based authentication. *J. Inf. Sec. Appl.* 19, 1 (2014), 25–44.
[7] Jan Camenisch, Maria Dubovitskaya, and Alfredo Rial. 2016. UC Commitments for Modular Protocol Design and Applications to Revocation and Attribute Tokens. In *CRYPTO 2016, Part III (LNCS)*, Matthew Robshaw and Jonathan Katz (Eds.), Vol. 9816. Springer, Heidelberg, 208–239. https://doi.org/10.1007/978-3-662-53015-3_8
[8] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. 2016. Universal Composition with Responsive Environments. In *ASIACRYPT 2016, Part II (LNCS)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.), Vol. 10032. Springer, Heidelberg, 807–840. https://doi.org/10.1007/978-3-662-53890-6_27
[9] Jan Camenisch, Aggelos Kiayias, and Moti Yung. 2009. On the Portability of Generalized Schnorr Proofs. In *EUROCRYPT 2009 (LNCS)*, Antoine Joux (Ed.), Vol. 5479. Springer, Heidelberg, 425–442.
[10] Jan Camenisch and Anna Lysyanskaya. 2004. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *CRYPTO 2004 (LNCS)*, Matthew Franklin (Ed.), Vol. 3152. Springer, Heidelberg, 56–72.
[11] Jan Camenisch and Victor Shoup. 2003. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *CRYPTO 2003 (LNCS)*, Dan Boneh (Ed.), Vol. 2729. Springer, Heidelberg, 126–144.
[12] Jan Camenisch and Markus Stadler. 1997. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *CRYPTO'97 (LNCS)*, Burton S. Kaliski Jr. (Ed.), Vol. 1294. Springer, Heidelberg, 410–424.
[13] Ran Canetti. 2000. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067. (2000). http://eprint.iacr.org/2000/067.
[14] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*. IEEE Computer Society Press, 136–145.
[15] Ran Canetti. 2004. Universally Composable Signature, Certification, and Authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*. IEEE Computer Society, 219. https://doi.org/10.1109/CSFW.2004.24
[16] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. 2012. Malleable Proof Systems and Applications. In *EUROCRYPT 2012 (LNCS)*, David Pointcheval and Thomas Johansson (Eds.), Vol. 7237. Springer, Heidelberg, 281–300.
[17] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. 2013. Malleable Signatures: Complex Unary Transformations and Delegatable Anonymous Credentials. Cryptology ePrint Archive, Report 2013/179. (2013). http://eprint.iacr.org/2013/179.
[18] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. 2013. Succinct Malleable NIZKs and an Application to Compact Shuffles. In *TCC 2013 (LNCS)*, Amit Sahai (Ed.), Vol. 7785. Springer, Heidelberg, 100–119. https://doi.org/10.1007/978-3-642-36594-2_6
[19] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. 2014. Malleable Signatures: New Definitions and Delegatable Anonymous Credentials. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*. IEEE Computer Society, 199–213. https://doi.org/10.1109/CSF.2014.22
[20] Melissa Chase and Anna Lysyanskaya. 2006. On Signatures of Knowledge. In *CRYPTO 2006 (LNCS)*, Cynthia Dwork (Ed.), Vol. 4117. Springer, Heidelberg, 78–96.
[21] David Chaum. 1982. Blind Signatures for Untraceable Payments. In *CRYPTO'82*, David Chaum, Ronald L. Rivest, and Alan T. Sherman (Eds.). Plenum Press, New York, USA, 199–203.
[22] David Chaum, Claude Crépeau, and Ivan Damgård. 1988. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *20th ACM STOC*. ACM Press, 11–19.
[23] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. 2007. Implementing Cryptographic Pairings over Barreto-Naehrig Curves (Invited Talk). In *PAIRING 2007 (LNCS)*, Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto (Eds.), Vol. 4575. Springer, Heidelberg, 197–207.
[24] Uriel Feige, Amos Fiat, and Adi Shamir. 1988. Zero-Knowledge Proofs of Identity. *Journal of Cryptology* 1, 2 (1988), 77–94.
[25] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86 (LNCS)*, Andrew M. Odlyzko (Ed.), Vol. 263. Springer, Heidelberg, 186–194.
[26] Georg Fuchsbauer. 2011. Commuting Signatures and Verifiable Encryption. In *EUROCRYPT 2011 (LNCS)*, Kenneth G. Paterson (Ed.), Vol. 6632. Springer, Heidelberg, 224–245.
[27] Georg Fuchsbauer and David Pointcheval. 2009. Formal to Practical Security. Springer-Verlag, Berlin, Heidelberg, Chapter Anonymous Consecutive Delegation of Signing Rights: Unifying Group and Proxy Signatures, 95–115. https://doi.org/10.1007/978-3-642-02002-5_6
[28] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *17th ACM STOC*. ACM Press, 291–304.
[29] Jens Groth. 2015. Efficient Fully Structure-Preserving Signatures for Large Messages. In *ASIACRYPT 2015, Part I (LNCS)*, Tetsu Iwata and Jung Hee Cheon (Eds.), Vol. 9452. Springer, Heidelberg, 239–259. https://doi.org/10.1007/978-3-662-48797-6_11
[30] Dennis Hofheinz and Victor Shoup. 2015. GNUC: A New Universal Composability Framework. *Journal of Cryptology* 28, 3 (July 2015), 423–508. https://doi.org/10.1007/s00145-013-9160-y
[31] Ralf Kuesters and Max Tuengerthal. 2013. The IITM Model: a Simple and Expressive Model for Universal Composability. Cryptology ePrint Archive, Report 2013/025. (2013). http://eprint.iacr.org/2013/025.
[32] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[33] Birgit Pfitzmann and Michael Waidner. 2000. Composition and Integrity Preservation of Secure Reactive Systems. In *ACM CCS 00*, S. Jajodia and P. Samarati (Eds.). ACM Press, 245–254.
[34] Claus-Peter Schnorr. 1990. Efficient Identification and Signatures for Smart Cards. In *CRYPTO'89 (LNCS)*, Gilles Brassard (Ed.), Vol. 435. Springer, Heidelberg, 239–252.
[35] Mårten Trolin and Douglas Wikström. 2005. Hierarchical Group Signatures. In *ICALP 2005 (LNCS)*, Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung (Eds.), Vol. 3580. Springer, Heidelberg, 446–458.

A SECURITY PROOF

We now prove Theorem 4.1. We have to prove that our scheme realizes $\mathcal{F}_{\text{dac}}$, which means proving that for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for every environment $\mathcal{E}$ we have $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{E}} \approx \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{E}}$.

To show that no environment $\mathcal{E}$ can distinguish the real world, in which it is working with Π_{dac} and adversary $\mathcal{A}$, from the ideal world, in which it uses $\mathcal{F}_{\text{dac}}$ with simulator $\mathcal{S}$, we use a sequence of games. We start with the real world protocol execution. In the next game we construct one entity $\mathcal{C}$ that runs the real world protocol for all honest parties. Then we split $\mathcal{C}$ into two pieces, a functionality $\mathcal{F}$ and a simulator $\mathcal{S}$, where $\mathcal{F}$ receives all inputs from honest parties and sends the outputs to honest parties. We start with a dummy functionality, and gradually change $\mathcal{F}$ and update $\mathcal{S}$ accordingly, to end up with the full $\mathcal{F}_{\text{dac}}$ and a satisfying simulator. First we define all intermediate functionalities and simulators, and then we prove that they are all indistinguishable from each other.

Game 1: This is the real world.

Game 2: We let the simulator $\mathcal{S}$ receive all inputs and generate all outputs by simulating the honest parties honestly. It also simulates the hybrid functionalities honestly. Clearly, this is equal to the real world.

Game 3: We now start creating a functionality $\mathcal{F}$ that receives inputs from honest parties and generates the outputs for honest parties. It works together with a simulator $\mathcal{S}$. In this game, we simply let $\mathcal{F}$ forward all inputs to $\mathcal{S}$, who acts as before. When $\mathcal{S}$ would generate an output, it first forwards it to $\mathcal{F}$, who then outputs it. This game hop simply restructures Game 2, we have Game 3 = Game 2.

Game 4: $\mathcal{F}$ now handles the setup queries, and lets $\mathcal{S}$ enter algorithms that $\mathcal{F}$ will store. $\mathcal{F}$ checks the structure of *sid*, and aborts if it does not have the expected structure. This does not change the view of $\mathcal{E}$, as $\mathcal{I}$ in the protocol performs the same check, giving Game 4 = Game 3.

Game 5: $\mathcal{F}$ now handles the verification queries using the algorithm that $\mathcal{S}$ defined in Game 4. In Game 4, $\mathcal{S}$ defined the Ver algorithm as the real world verification algorithm so we have Game 5 = Game 4.

Game 6: $\mathcal{F}$ now also handles the delegation queries. If both the delegator and the delegatee are honest, $\mathcal{S}$ does not learn the attribute values and must simulate the real world protocol with dummy values. As all communication is over a secure channel, this difference is not noticable by the adversary.

If the delegatee is corrupt, $\mathcal{S}$ learns the attribute values $\mathcal{S}$ can simulate the real world protocol with the correct input. If the delegator is corrupt and the delegatee honest, $\mathcal{S}$ has to take more care: The corrupt delegator may have received delegated credentials from other corrupt users, without $\mathcal{S}$ and $\mathcal{F}$ knowing. If $\mathcal{S}$ would make a delegation query with $\mathcal{F}$ on the delegator's behalf, $\mathcal{F}$ would reject as it does not possess the required attributes for this delegation, invalidating the simulation. In this case, $\mathcal{S}$ first informs $\mathcal{F}$ of the missing delegations, such that $\mathcal{F}$'s records accept the delegation, and only then calls $\mathcal{F}$ on the delegator's behalf for this delegation.

As $\mathcal{S}$ only lacks information to simulate when both parties are honest, but this change is not noticable due to the use of a secure channel, Game 6 ≈ Game 5.

Game 7: $\mathcal{F}$ now generates the attribute tokens for honest parties, using the Present algorithm that $\mathcal{S}$ defined in Game 4. First, $\mathcal{F}$ checks whether the party is eligible to create such an attribute token, and aborts otherwise. This does not change $\mathcal{E}$'s view, as the real world protocol performs an equivalent check. Second, $\mathcal{F}$ tests whether attribute token *at* generated with Present is valid w.r.t. Ver before outputting *at*. $\mathcal{S}$ defined Present to sign a valid witness for the NIZK that *at* is, and Ver will verify the NIZK. By completeness of all the sibling signature schemes Sib and completeness of NIZK, *at* will be accepted by Ver. This shows that $\mathcal{F}$ outputs an attribute token if and only if the real world party would output an attribute token.

Next, we must show that the generated attribute token is indistinguishable between the real and ideal world. Both the real world protocol and the Present algorithm compute

$$at \leftarrow \mathsf{NIZK}\Big\{(\sigma_1,\ldots,\sigma_L,cpk_1,\ldots,cpk_L,\langle a'_{i,j}\rangle_{i\notin D},tag):\\ \bigwedge_{i=1}^{L} 1=\mathsf{Sib}_{i-1}.\mathsf{Verify}_1(cpk_{i-1},\sigma_i,cpk_i,a'_{i,1},\ldots,a'_{i,n_i})\\ \wedge 1=\mathsf{Sib}.\mathsf{Verify}_2(cpk_L,tag,m)\Big\}$$

but in the real world, a party uses his own credential every time he proves this statement, and $\mathcal{F}$ creates a fresh credential for every signature. Note that the credential only concerns the witness of the zero-knowledge proof. By the witness indistinguishability of the zero-knowledge proofs, this change is not noticable and we have Game 7 ≈ Game 6.

Game 8: $\mathcal{F}$ now guarantees unforgeability of attribute tokens. We make this change gradually, where in the first intermediate game we guarantee unforgeability of level 1 attribute tokens, then of level 2, and so forth, and we prove that each game is indistinguishable from the previous.

If the unforgeability check for level L credentials triggers with non-negligible probability, there must be an attribute token *at* that was valid before but is rejected by the unforgeability check of $\mathcal{F}$. This means that one of the two statements must hold with non-negligible probability:

- *at* proves knowledge either of a public key cpk_L that belongs to an honest user with the correct attributes, but this user never signed m (as otherwise the unforgeability check would not trigger)
- *at* proves knowledge of a public key cpk_L that does not belong to an honest user.

In the first case, we can reduce to the unforgeability-2 property of Sib: There can only be polynomially many delegations of a level L credential to an honest user. Pick a random one and simulate the receiving party with the public key vk as received from the unforgeability game of Sib. When the user delegates this credential, use the Sign_1 oracle, and when presenting the credential, use the Sign_2 oracle. Finally, when $\mathcal{F}$ sees an attribute token *at* that it considers a forgery, the soundness of NIZK allows us to extract from the zero-knowledge proof. With non-negligible probability, $cpk_L = vk$, and then *tag* is a Sib forgery.

In the second case, we can reduce to the unforgeability-1 property of Sib: If $L = 1$, simulate the issuer with $ipk \leftarrow vk$, where vk is taken from the Sib unforgeability game. As *isk* us not known to the simulator, we simulate π_{isk}, and define the Present algorithm to simulate the proof such that the issuer secret key is not needed. $\mathcal{I}$ uses the Sign_1 oracle to delegate. If a delegation was chosen, simulate the receiver using $cpk_i \leftarrow vk$. If $L > 1$, there can only be polynomially many delegations that give an honest user a credential of level $L-1$. Pick a random one and simulate the receiving party with $cpk_{L-1} \leftarrow vk$. Use the Sign_1 oracle to delegate this credential, and the Sign_2 oracle to present this credential. Finally, when $\mathcal{F}$ sees an attribute token *at* that it considers a forgery, extract from the zero-knowledge proof. With non-negligible probability, $cpk_{L-1} = vk$, and then σ_L is a Sib forgery on message cpk_L.

We provide the detailed description of the simulator on Fig. 6. $\mathcal{F}$ of Game 8 of equal to $\mathcal{F}_{\mathsf{dac}}$, conclusing our sequence of games. □

Setup

Honest $\mathcal{I}$

- On input (SETUP, *sid*, $\langle n_i \rangle_i$) from $\mathcal{F}$.
  - Parse *sid* as ($\mathcal{I}$, *sid'*) and give "$\mathcal{I}$" input (SETUP, *sid*, $\langle n_i \rangle_i$).
  - When "$\mathcal{I}$" outputs (SETUPDONE, *sid*), $\mathcal{S}$ takes its public key *ipk* and secret key *isk* and defines Present and Ver, and the attribute spaces $\langle \mathbb{A}_i \rangle_i$.
    - Define Present(m, $\vec{a}_1, \ldots, \vec{a}_L$) as follows: Run ($cpk_i$, csk_i) $\leftarrow$ $\mathrm{Sig}_i.\mathrm{Gen}(1^\kappa)$ for $i = 1, \ldots, L$. Compute $\sigma_1 \leftarrow \mathrm{Sig}_0.\mathrm{Sign}(isk; cpk_1, \vec{a}_1)$ and $\sigma_i \leftarrow \mathrm{Sig}_{i-1}.\mathrm{Sign}(csk_{i-1}, cpk_i, \vec{a}_i)$ for $i = 2, \ldots, L$. Next, compute *at* as in the real world protocol and return *at*.
    - Define $\mathrm{Ver}(at, m, \vec{a}_1, \ldots, \vec{a}_L)$ as the real world verification algorithm that verifies with respect to *ipk*.
    - Define $\mathbb{A}_i$ as $\mathbb{G}_1$ for odd i and as $\mathbb{G}_2$ for even i.
  - $\mathcal{S}$ sends (SETUP, *sid*, Present, Ver, $\langle \mathbb{A}_i \rangle_i$) to $\mathcal{F}$.

Corrupt $\mathcal{I}$

- $\mathcal{S}$ notices this setup as it notices $\mathcal{I}$ registering a public key with "$\mathcal{F}_{\mathrm{ca}}$" with *sid* = ($\mathcal{I}$, *sid'*).
  - If the registered key is of the form (*ipk*, π_{isk}) and π_{isk} is valid, $\mathcal{S}$ extracts *isk* from π_{isk}.
  - $\mathcal{S}$ defines Present, Ver and $\langle \mathbb{A}_i \rangle$ as when $\mathcal{I}$ is honest, but now depending on the extracted key.
  - $\mathcal{S}$ sends (SETUP, *sid*) to $\mathcal{F}$ on behalf of $\mathcal{I}$.
- On input (SETUP, *sid*) from $\mathcal{F}$.
  - $\mathcal{S}$ sends (SETUP, *sid*, Present, Ver, $\langle \mathbb{A}_i \rangle_i$) to $\mathcal{F}$.
- On input (SETUPDONE, *sid*) from $\mathcal{F}$
  - $\mathcal{S}$ continues simulating "$\mathcal{I}$".

Delegate

Honest $\mathcal{P}$, $\mathcal{P}'$

- $\mathcal{S}$ notices this delegation as it receives (ALLOWDEL, *sid*, *ssid*, $\mathcal{P}$, $\mathcal{P}'$, L) from $\mathcal{F}$.
  - $\mathcal{S}$ picks dummy attribute values $\vec{a}_1, \ldots, \vec{a}_L$ and gives "$\mathcal{P}$" input (DELEGATE, *sid*, *ssid*, $\vec{a}_1, \ldots, \vec{a}_L$, $\mathcal{P}'$).
  - When "$\mathcal{P}'$" outputs (DELEGATE, *sid*, *ssid*, $\vec{a}_1, \ldots, \vec{a}_L$, $\mathcal{P}$), output (ALLOWDEL, *sid*, *ssid*) to $\mathcal{F}$.

Honest $\mathcal{P}$, corrupt $\mathcal{P}'$

- $\mathcal{S}$ notices this delegation as it receives (ALLOWDEL, *sid*, *ssid*, $\mathcal{P}$, $\mathcal{P}'$, L) from $\mathcal{F}$.
  - Output (ALLOWDEL, *sid*, *ssid*) to $\mathcal{F}$.
- $\mathcal{S}$ receives (DELEGATE, *sid*, *ssid*, $\vec{a}_1, \ldots, \vec{a}_L$, $\mathcal{P}$) as $\mathcal{P}'$ is corrupt.
  - $\mathcal{S}$ gives "$\mathcal{P}$" input (DELEGATE, *sid*, *ssid*, $\vec{a}_1, \ldots, \vec{a}_L$, $\mathcal{P}'$).

Honest $\mathcal{P}'$, corrupt $\mathcal{P}$

- $\mathcal{S}$ notices this delegation as "$\mathcal{P}$" outputs (DELEGATE, *sid*, *ssid*, $\vec{a}_1, \ldots, \vec{a}_L$, $\mathcal{P}$).
  - If $L > 1$ and $\mathcal{S}$ has not simulated delegating attributes $\vec{a}_1, \ldots, \vec{a_{L-1}}$ to $\mathcal{P}$, and there is a corrupt party $\mathcal{P}''$ that has attributes $\vec{a}_1, \ldots, \vec{a}_i$ for $0 < i < L-1$ (note that if the root delegator $\mathcal{I}$ is corrupt, $i = 0$), $\mathcal{P}''$ may have delegated $\vec{a}_i, \ldots, \vec{a}_{L-1}$ to $\mathcal{P}'$ without $\mathcal{S}$ noticing. Therefore, $\mathcal{S}$ needs to delegate attributes $\vec{a}_i, \ldots, \vec{a}_{L-1}$ in the ideal world, which is possible as $\mathcal{P}''$ is corrupt: $\mathcal{S}$ sends (DELEGATE, *sid*, *ssid*, $\vec{a}_1, \ldots, \vec{a}_i$, $\mathcal{P}'$) on $\mathcal{P}''$'s behalf to $\mathcal{F}$ and allows the delegation, and for $j = i+1, \ldots, L-1$, sends (DELEGATE, *sid*, *ssid*, $\vec{a}_1, \ldots, \vec{a}_j$, $\mathcal{P}'$) on $\mathcal{P}''$'s behalf to $\mathcal{F}$, allowing every delegation. Note that $\mathcal{P}'$ now possesses attributes $\vec{a}_1, \ldots, \vec{a_{L-1}}$ in $\mathcal{F}$'s records.
  - Send (DELEGATE, *sid*, *ssid*, $\vec{a}_1, \ldots, \vec{a}_L$, $\mathcal{P}'$) on $\mathcal{P}$'s behalf to $\mathcal{F}$.
- On input (ALLOWDEL, *sid*, *ssid*, $\mathcal{P}$, $\mathcal{P}'$, L) from $\mathcal{F}$.
  - Output (ALLOWDEL, *sid*, *ssid*) to $\mathcal{F}$.

Corrupt $\mathcal{P}$, $\mathcal{P}$'

Nothing to simulate.

Present

Nothing to simulate.

Verify

Nothing to simulate.

Figure 6: Description of the Simulator