Delegatable Attribute-based Anonymous Credentials from Dynamically Malleable Signatures

Johannes Blömer and Jan Bobolz
Paderborn University
{bloemer, jan.bobolz}@uni-paderborn.de

Abstract

In this paper, we introduce the notion of *delegatable attribute-based* anonymous credentials (DAAC). Such systems offer fine-grained anonymous access control and they give the credential holder the ability to issue more restricted credentials to other users. In our model, credentials are parameterized with attributes that (1) express what the credential holder himself has been certified and (2) define which attributes he may issue to others. Furthermore, we present a practical construction of DAAC. For this construction, we deviate from the usual approach of embedding a certificate chain in the credential. Instead, we introduce a novel approach for which we identify a new primitive we call *dynamically malleable signatures* (DMS) as the main ingredient. This primitive may be of independent interest. We also give a first instantiation of DMS with efficient protocols.

Keywords: delegatable credentials, anonymous credentials, attribute-based credentials, authentication, malleable signatures

1 Introduction

In this paper, we construct delegatable attribute-based anonymous credentials (DAAC) that offer fine-grained anonymous access control for many typical scenarios. For example, consider a company with the usual hierarchical organization structure. We want the company owner to be able to grant appropriate access rights to department managers. For this, he issues each of them a *credential* with certain attributes encoding what rights the department manager has. The department managers, in turn, should be able to grant appropriate subsets of their rights to their staff by *delegating* a (weaker) version of their credential to them. Then a staff member may want to grant access rights to her interns, etc.

This scenario could be trivially realized using certificate chains (similar to the ones used in TLS): the company owner signs the public key of a department manager alongside some string that encodes which attributes the manager has and which ones he may delegate to his staff. Then the department manager can extend the chain by signing a staff member's public key, and so on.

However, our goal is to enable *anonymity* for authentication: An authorized user requesting access to a resource should be indistinguishable from *all other users who have access to it.* Still,

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901).

the verifier, who is executing the authentication checks, should be assured that only authorized users can pass them. Overall, we have the following requirements:

R1 The verifier must not learn the sequence of delegations a user's credential went through.

R2 The verifier must only learn as much as necessary about the attributes of the authenticating user or of any users involved in the delegation.

R3 A user must not be able to grant other users more rights/attributes than he is allowed to.

R4 A user should not be able to pass the authentication checks without being given an appropriate credential.

R5 Users shall remain anonymous while delegating and receiving a credential.

Most previous constructions of delegatable credentials [BCC+09, AN11, Fuc10] fulfill R1, R3, R4, and R5. However, in those constructions the attributes of all users in the chain are presented in plain to the verifier. This violates R2.

A more recent scheme [CDD17] supports R1, R3, R4, and R2. However, in their construction, credential holders see all attributes of the users in the delegation chain. This violates R5.

In this paper, we introduce the first construction of practical delegatable attribute-based anonymous credentials that supports all five requirements.

Our model of delegatable attribute-based anonymous credentials. In DAAC, users have a single secret key and can derive an arbitrary number of unlinkable public keys (usually called *pseudonyms*). Users can issue credentials to other users (also anonymously, if desired). A credential is parameterized with a delegation flag $d \in \{0, 1\}$, which determines whether or not the credential is delegatable, and an attribute vector $(A_1, \dots, A_n) \in (\mathbb{A} \cup \{\star\})^n$ for some set $\mathbb{A}$ (e.g., $\mathbb{A} = \mathbb{Z}_p$). Each A_i in the attribute vector either takes on a concrete value from $\mathbb{A}$, or the special value $\star$, which can be interpreted as a placeholder that can be replaced with an arbitrary $\mathbb{A}$-value.

We define the *covers* relation on attribute vectors that will determine what attributes the user can delegate and show. A vector $(A_1, \dots, A_n)$ *covers* another vector $(A'_1, \dots, A'_n)$ (we write $\vec{A} \succeq \vec{A}'$) if and only if $(A_i \neq \star) \Rightarrow (A'_i = A_i)$ for all i. This means that the placeholder $\star$ can be replaced with any concrete $\mathbb{A}$ value (or $\star$ again), whereas concrete values from $\mathbb{A}$ cannot be changed.

Given a credential with delegation flag $d = 1$ and attribute-vector $\vec{A}$, a user is able to issue a credential with any delegation flag $d^* \in \{0, 1\}$ and any attributes $\vec{A}^*$ as long as $\vec{A} \succeq \vec{A}^*$. A credential with delegation flag $d = 0$ cannot be delegated any further. When *showing* a credential for an access policy ϕ (e.g., a Boolean formula over statements like "$A_2 = 13$"), the user proves that his attributes cover some concrete $\vec{A}' \in \mathbb{A}^n$ for which $\phi(\vec{A}') = 1$. Note that it is natural that whatever users can delegate, they can also show.

In the simplest case, each attribute A_i may just encode a Boolean access right and users can delegate subsets of their access rights. In the concrete instantiation based on Construction 5.1 in our paper, you can encode arbitrary elements of $\mathbb{A} = \mathbb{Z}_p$ (e.g., short strings) into credentials, hence our scheme can realize more elaborate authentication scenarios. As a small example for this, the state of California may issue San Francisco a delegatable credential like $\vec{A} = (\star, \star, \texttt{San Francisco})$. This allows the city to delegate credentials with attributes $\vec{A}^* = (\texttt{John}, \texttt{Doe}, \texttt{San Francisco})$ to its citizens, but prohibits issuing credentials encoding other cities.

When authenticating with a credential, the only information revealed to the verifier is the root of the delegation chain (e.g., the company owner or the state), a pseudonym of the authenticating user, and the fact that the credential's attributes fulfill some predicate.

Idea of our construction. We deviate from the usual way of constructing delegatable credentials and instead follow a novel approach. We identify a new primitive we call *dynamically malleable signatures* (DMS) as the main ingredient. DMS are similar to usual malleable signatures, but the set of allowed transformations is specific to each signature and can be incrementally restricted over time ("*dynamic*" refers to the fact that the set of allowed transformations is not static but can be changed "at runtime" for each signature). More specifically, the Sign algorithm takes some vector of messages $(m_1, \dots, m_n)$ and an index set $I \subseteq \{1, \dots, n\}$ as input and produces a signature σ and a *malleability key* mk. The index set I determines which of the positions in the vector are malleable, i.e. given σ and mk, anyone can derive a signature σ' on any message $(m'_1, \dots, m'_n)$ as long as $m'_i = m_i$ for all $i \notin I$. This process also yields a malleability key mk' for σ', which can be restricted to allow further modification of σ' only on some (smaller) index set $I' \subseteq I$. In Section 5, we give an efficient construction of DMS with supporting protocols. Our construction is based on the Pointcheval-Sanders signature scheme [PS16] and it can be proven secure in the generic group model.

Using any secure DMS scheme with an efficient protocol to derive a signature on a committed value, we generically implement a DAAC as follows: With some details omitted, a credential for a user with secret usk and attributes $\vec{A} = (15, 7, \star, \star)$ is a dynamically malleable signature on usk and $\vec{A}$. For each $\star$ in $\vec{A}$, we instead sign 0. More formally, in this example we would sign $(m_1, \dots, m_5) := (usk, 15, 7, 0, 0)$ and allow the receiver to use malleability on the first index (to change usk when delegating) and the last two indices (to model the $\star$), i.e. $I = \{1, 4, 5\}$. Unforgeability of the signature scheme then guarantees that this user cannot produce a credential whose first two attributes are not 15 and 7 (cf. requirement R3). If he wants to delegate attributes $\vec{A}' = (15, 7, 13, \star)$ to another user with secret key usk', the two parties engage in a protocol to derive a signature on $(usk', 15, 7, 13, 0)$ such that only the first and the last message can be changed further, i.e. $I' = \{1, 5\} \subset I$. Note that the issuer's usk or his exact attributes are not part of the derived credential, immediately implying R1. Our delegation protocol will also ensure R5. To mark the credential non-delegatable ($d = 0$), the delegator can remove the first index from the index set, which precludes the receiver from changing the signature to any other secret key $usk'' \neq usk'$. Showing the credential to a verifier follows standard procedure [Lys02], i.e. the user runs a zero-knowledge protocol to prove knowledge of a signature on his user secret and on attributes fulfilling some policy (and that his user secret is consistent with his pseudonym). The zero-knowledge property ensures requirement R2 while the proof of knowledge property and the unforgeability of the signature scheme ensures R4.

Related work on delegatable credentials. Chase and Lysyanskaya introduced the first anonymous delegatable credential system [CL06], which extended the idea of anonymous credentials [Cha85] to enable delegation. Later, Belenkiy et al. published an elegant construction of delegatable credentials [BCC$^+$09] and introduced formal security requirements. In their paper, delegatable credentials are defined through *levels*. Any user can publish a pseudonym pk_{root} and issue a level $L = 1$ credential to another user. Then, a level L credential can be used to derive a level $L + 1$ credential for another user. When showing a credential, the verifier learns the root pseudonym pk_{root} of the credential, the prover's pseudonym pk_L, and the credential's level L.

The construction of [BCC$^+$09] allows users to attach public attributes to a credential when delegating, meaning that a level L credential is parameterized with L attribute descriptions

$(A_1, \ldots, A_L) \in (\{0,1\}^*)^L$, where the issuer of the level $\ell \leq L$ credential chooses A_ℓ. However, they do not describe a way to hide the attributes of any user in the delegation chain, which weakens anonymity considerably (cf. our requirement R2). Furthermore, there are no restrictions on the attribute strings a delegator can add when delegating a credential (cf. requirement R3). Hence the burden of verifying plausibility of delegated attributes lies with the verifier. For example, after seeing a credential with attributes ("manager of IT department", "programmer in IT department"), the verifier needs to do a "sanity check" whether or not a manager may actually issue a credential to a programmer. In contrast, the verifier in our DAAC does not need to do this check (in fact, he cannot do it as he does not get to see the attributes). Instead, the controlled malleability property of the underlying signature scheme guarantees that a credential's attributes can be legitimately derived throughout the delegation steps. This shifts the responsibility of defining what delegations are allowed from the verifier to the issuers.

If we instantiate our generic construction of DAAC with our concrete DMS scheme (Section 5), a credential with n attributes consists of at most $n+3$ group elements. In particular, the credential size is independent of the delegation chain length. Using standard variants of Schnorr's protocol, showing the credential can be done very efficiently compared to [BCC+09], whose credentials are Groth-Sahai proofs with size linear in the chain length. As a trade-off, we reach this level of efficiency mainly by (1) not encoding the delegation chain into credentials (which also precludes the feature of tracing the sequence of credential owners using a trapdoor), and (2) leveraging a new ad-hoc algebraic assumption for the construction of our concrete DMS scheme (there *are* delegatable credentials that are secure under standard assumptions, e.g., DLIN [CKLM14]).

Most other constructions [AN11, CKLM14, Fuc10] also follow roughly the same techniques as [BCC+09], i.e. using malleable proof systems (like Groth-Sahai) as a building block, improving upon and generalizing the original idea. However, there do not seem to be any constructions that improve upon their handling of (public) attributes.

More recently, Camenisch et al. published a delegatable credential system [CDD17]. Their construction is very efficient and practical. They achieve this by allowing credential holders to see all attributes on all levels in plain, i.e. not offering anonymity for delegators. In many contexts, this is not a problem. However, consider the example of a distributed car sharing scenario where the car owner is issued a credential for his car. In a car-sharing fashion, he can delegate his access rights to any other person. In this scenario, the car owner has no reason to reveal his identity. Our construction shows that one does not have to sacrifice support for such scenarios to achieve practicality: Namely, our scheme's efficiency is comparable to [CDD17] while offering anonymity for delegators (R5).

Related work on malleable signatures. Malleable signature schemes allow anyone to transform a signature on a message m to be valid for a different message m' for a well-defined set $\mathcal{T}$ of allowed transformations on m (e.g., [CKLM14]).

In contrast, our notion of DMS allows signers to choose transformation sets $\mathcal{T}_\sigma$ on a per-signature basis, which can be further restricted to some subset $\mathcal{T}_{\sigma'}$ when transforming σ to σ'.

The general idea for DMS is similar to homomorphic signatures like [BFKW09, ABC+15, Fre12, AL11]. In these constructions (mostly designed for network coding), a signer effectively signs a vector space by signing its base vectors with a homomorphic signature scheme. This allows computing signatures on any vector in the vector space. Like DMS, homomorphic signature schemes allow to derive signatures on related messages. However, the main feature of DMS is that one can derive signatures that are *more restricted* than the original. Furthermore, one cannot combine two restricted signatures to produce a signature on a message not covered by either of them.

Structure of this paper. We introduce notation and basic definitions in Section 2. In Section 3, we introduce the formal definition for delegatable attribute-based anonymous credentials and explain how to utilize them in practice. Then, in Section 4, we define DMS and supporting protocols. In Section 5 we give a concrete construction of DMS with an efficient protocol. We instantiate delegatable attribute-based anonymous credentials in Section 6 by giving a generic construction from DMS with efficient protocols. Finally, we conclude the paper in Section 7 with some notes on future work.

2 Basics and notation

For a random variable X, $[X] := \{x \mid \Pr[X = x] > 0\}$ is the *support* of X. With $X \leftarrow S$, we denote that X is chosen uniformly at random from the set S. If X and Y are identically distributed random variables, we write $X \approx Y$. With $X \leftarrow A(y)$ we denote that X is generated by running the probabilistic algorithm A on input y. The notation $\Pr[X_1 \leftarrow S, X_2 \leftarrow A(y, X_1); \phi(X_1, X_2)]$ denotes the probability that the predicate $\phi(X_1, X_2)$ holds in the probability space described by X_1, X_2. For a prime number p, $\mathbb{Z}_p$ is the field of order p and $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$.

Definition 2.1. Let A, B be probabilistic interactive algorithms that halt on every input. We write $y_A \leftarrow A(x_A) \leftrightarrow B(x_B) \rightarrow y_B$ to denote that A on input x_A interacts with B on input x_B; then A outputs y_A and B outputs y_B. Furthermore, we define $\text{output}_A[A(x_A) \leftrightarrow B(x_B)]$ to be the random variable taking on y_A, i.e. the output of A after interacting with B. $\diamond$

Definition 2.2 (Protocols and signatures of knowledge). The expression $ZKAK[(w); (x, w) \in \Psi]$ denotes a zero-knowledge argument of knowledge for the relation Ψ. $NIZK[(w); (x, w) \in \Psi](m)$ denotes a *signature of knowledge* on message m for the relation Ψ. $\diamond$

Zero-knowledge arguments of knowledge can be implemented, for example, using Damgård's technique [Dam00] on Schnorr-like Σ protocols. Signatures of knowledge can be implemented, for example, using the Fiat-Shamir heuristic.

3 Delegatable attribute-based anonymous credentials

In this section, we define DAAC. Each credential is parameterized with a vector $\vec{A} = (A_1, \ldots, A_n) \in (\mathbb{A} \cup \{\star\})^n$, a delegation flag d, and the root authority's pseudonym pk_{root}.

To define what a user may do with his credential, we need the relation "*covers*". An attribute vector $\vec{A} = (A_1, \ldots, A_n) \in (\mathbb{A} \cup \{\star\})^n$ *covers* another attribute vector $\vec{A}' = (A'_1, \ldots, A'_n) \in (\mathbb{A} \cup \{\star\})^n$ if $(A_i \neq \star) \Rightarrow (A'_i = A_i)$ for all $1 \leq i \leq n$. In this case we write $\vec{A} \succeq \vec{A}'$. Furthermore, we say that an attribute vector $\vec{A} \in (\mathbb{A} \cup \{\star\})^n$ *covers* a predicate $\phi : \mathbb{A}^n \rightarrow \{0, 1\}$ if it covers some vector without $\star$ fulfilling ϕ, i.e. $\exists \vec{A}' \in \mathbb{A}^n : \ \vec{A} \succeq \vec{A}' \ \wedge \ \phi(\vec{A}') = 1$. We write $\vec{A} \succeq \phi$.

Let *cred* be a credential with attributes $\vec{A}$, delegation flag d, and root authority's pseudonym pk_{root} (we say that *cred* is *rooted* at pk_{root}). With *cred*, the user can do the following: (1) Prove possession of a pk_{root}-rooted credential that covers some predicate ϕ, and (2) if $d = 1$, he can issue a derived credential still rooted at pk_{root} with attributes $\vec{A}' \in (\mathbb{A} \cup \{\star\})^n$ iff $\vec{A} \succeq \vec{A}'$.

3.1 Formal definition

Definition 3.1. A DAAC system consists of the following ppt algorithms:

$\text{Setup}(1^\lambda) \rightarrow (pp, osk)$ generates public parameters pp and an opening key osk. We assume an attribute universe $\mathbb{A}$ be to be encoded in pp.

$\mathrm{KeyGen}(pp) \to (usk, id)$ generates a user secret usk and an identity id.

$\mathrm{FormNym}(pp, usk, 1^n) \to (pk, sk)$ generates a pseudonym pk and a pseudonym secret sk such that credentials rooted at pk support n attributes.

$\mathrm{Open}(pp, osk, pk) = id$ is a deterministic algorithm that extracts an identity id from the pseudonym pk.

$\mathrm{CreateCred}(pp, pk, sk) \to cred$ creates a root credential, i.e. a delegatable credential with attributes $(\star, \dots, \star)$ and delegation flag $d = 1$, rooted at pk.

$\mathrm{DelegIssue}(pp, pk_{\mathrm{root}}, usk, cred, \vec{A}^*, d^*, pk^*)$
$\leftrightarrow \mathrm{DelegRcv}(pp, pk_{\mathrm{root}}, \vec{A}^*, d^*, pk^*, sk^*, usk^*) \to cred^*$ is an interactive protocol with common input the root's pseudonym pk_{root}, the receiver's pseudonym pk^*, the attributes to be issued $\vec{A}^*$, and the delegation flag $d^* \in \{0, 1\}$. Additionally, the issuer gets his user secret usk as private input, as well as his credential $cred$. Finally, the receiver gets his pseudonym secret sk^* and user secret usk^* as private input. After the protocol, the receiver side outputs a credential $cred^*$ or the failure symbol $\perp$.

$\mathrm{ShowProve}(pp, pk_{\mathrm{root}}, pk, \phi, sk, usk, cred) \leftrightarrow \mathrm{ShowVrfy}(pp, pk_{\mathrm{root}}, pk, \phi) \to b$ is an interactive protocol with common input the root's pseudonym pk_{root}, the prover's pseudonym pk, and a statement over attributes $\phi : \mathbb{A}^n \to \{0, 1\}$. The prover gets his pseudonym secret sk, his user secret usk, and his credential $cred$ as private input. The verifier outputs a bit b.

Furthermore, we require three helper predicates that enable simpler correctness and security definitions: $\mathrm{CheckPseud}(pp, pk, sk, usk)$, $\mathrm{CheckShow}(pp, pk_{\mathrm{root}}, pk, \phi, sk, usk, cred)$, and $\mathrm{CheckDeleg}(pp, pk_{\mathrm{root}}, usk, cred, \vec{A}^*)$.

For correctness, we require that

- All pseudonyms (pk, sk) generated by $\mathrm{FormNym}(pp, usk, 1^n)$ pass the CheckPseud check. We call (pk, sk) that pass CheckPseud *valid*.
- For all $(usk, id) \in [\mathrm{KeyGen}(pp)]$ and valid (pk, sk): $\mathrm{Open}(pp, osk, pk) = id$.
- ShowVrfy $\leftrightarrow$ ShowProve succeeds if its input passes CheckShow and CheckPseud.
- In DelegIssue $\leftrightarrow$ DelegRcv, if the protocols' inputs pass CheckDeleg for the issuer's credential and CheckPseud for the receiver's pseudonym, then DelegRcv does not output the error symbol $\perp$.
- Any output of $\mathrm{DelegRcv}(pp, pk_{\mathrm{root}}, \vec{A}^*, d^*, pk^*, sk^*, usk^*)$ is either $\perp$ or a credential $cred^*$ that passes $\mathrm{CheckShow}(pp, pk_{\mathrm{root}}, pk', \phi, sk', usk^*, cred^*)$ for all $\vec{A}^* \succeq \phi$. If $d^* = 1$, it also passes $\mathrm{CheckDeleg}(pp, pk_{\mathrm{root}}, usk^*, cred^*, \vec{A}')$ for all $\vec{A}^* \succeq \vec{A}'$.
- Root credentials $cred \in [\mathrm{CreateCred}(pp, pk_{\mathrm{root}}, sk_{\mathrm{root}})]$ are universal, i.e. if (pk, sk) are valid, then $\mathrm{CheckShow}(pp, pk_{\mathrm{root}}, pk, \phi, sk, usk, cred) = 1$ for all satisfiable ϕ. Furthermore, $\mathrm{CheckDeleg}(pp, pk_{\mathrm{root}}, usk, cred, \vec{A}^*) = 1$ for all $\vec{A}^* \in (\mathbb{A} \cup \{\star\})^n$. $\diamond$

A more formal version of correctness can be found in Appendix D.

The system is set up using Setup, and the special opener secret osk is given to a trusted authority. Any user can join the system by simply calling KeyGen to generate their own user secret usk and identity id. With the user secret, one can generate any number of pseudonyms pk using FormNym. A user can declare himself a credential-issuing authority by publishing one of his pseudonyms pk_{root} and creating a root credential with CreateCred which allows him to delegate arbitrary credentials rooted at pk_{root}. To delegate a credential, the delegator runs DelegIssue while the receiver runs DelegRcv. To show a credential, a user runs ShowProve while the verifier runs ShowVrfy. In case of abuse, the opener secret *osk* can be used to extract the identity of a user from one of his pseudonyms.

Note the omission of a registration mechanism that prevents users from repeatedly generating ephemeral identities to circumvent persecution by the opener. A registration mechanism can be generically constructed from the credential system itself. We explain this in Section 3.2.

For security, we expect anonymity (users cannot be traced when showing, delegating, or receiving credentials) and soundness (users cannot impersonate other users or show credentials they have not been issued).

Definition 3.2 (Anonymity). A DAAC system Π has *anonymity* if there is an ppt algorithm $(pp, osk, td) \leftarrow \mathfrak{S}^{pp}(1^\lambda)$ whose output is such that (pp, osk) is distributed exactly like $\text{Setup}(1^\lambda)$. Furthermore, there are ppt simulators $\mathfrak{S}_{\text{ShowProve}}$,$\mathfrak{S}_{\text{DelegIssue}}$,$\mathfrak{S}_{\text{DelegRcv}}$ such that no (unrestricted) $\mathcal{A}$ can distinguish between interacting with

- $\text{ShowProve}(pp, pk_{\text{root}}, pk, \phi, sk, usk, cred)$ and $\mathfrak{S}_{\text{ShowProve}}(td, pk_{\text{root}}, pk, \phi)$
- $\text{DelegIssue}(pp, pk_{\text{root}}, usk, cred, \vec{A}^*, d^*, pk^*)$ and $\mathfrak{S}_{\text{DelegIssue}}(td, pk_{\text{root}}, \vec{A}^*, d^*, pk^*, usk_{\text{alt}}, cred_{\text{alt}})$ for *any* $usk_{\text{alt}}, cred_{\text{alt}}$ that pass the $\text{CheckDeleg}(pp, pk_{\text{root}}, usk_{\text{alt}}, cred_{\text{alt}}, \vec{A}^*)$ check.
- $\text{DelegRcv}(pp, pk_{\text{root}}, \vec{A}^*, d^*, pk^*, sk^*, usk^*)$ and $\mathfrak{S}_{\text{DelegRcv}}(td, pk_{\text{root}}, \vec{A}^*, d^*, pk^*)$

Furthermore, for all ppt $\mathcal{A}$ there is a negligible function *negl* s.t. for all $\lambda \in \mathbb{N}$,

$$\Pr[(pp, osk, td) \leftarrow \mathfrak{S}^{pp}(1^\lambda), (usk_0, id_0), (usk_1, id_1) \leftarrow \text{KeyGen}(pp), b \leftarrow \{0,1\},$$
$$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\text{FormNym}}(\cdot),\mathcal{O}_{\text{Open}}(\cdot)}(1^\lambda, pp, td, usk_0, id_0, usk_1, id_1); b = b'] \leq 1/2 + negl(\lambda)$$

where $\mathcal{O}_{\text{FormNym}}(1^n)$ returns pseudonyms of usk_b, i.e. it runs $(pk, sk) \leftarrow \text{FormNym}(pp, usk_b, 1^n)$ and returns pk. $\mathcal{O}_{\text{Open}}(pk)$ returns $\perp$ if pk was previously output by $\mathcal{O}_{\text{FormNym}}(\cdot)$, otherwise returns $\text{Open}(pp, osk, pk)$. ◇

A more formally rigorous definition can be found in Appendix D. Our simulators get as input the simulation trapdoor *td* and the common public input of the simulated protocol. In addition, $\mathfrak{S}_{\text{DelegIssue}}$ gets *any* $usk_{\text{alt}}, cred_{\text{alt}}$ (which can be completely uncorrelated to the actual delegator's *usk*, *cred*) as input to help with the simulation. The experiment in the last part of the definition models a situation where the ppt algorithm $\mathcal{A}$ knows all secrets except the opener's *osk*. He interacts with one of two possible honest users (who generated their *usk* honestly) and may request additional pseudonyms from that unknown user. Additionally, $\mathcal{A}$ may use usk_0, usk_1 to create any situation it wants for the two users. We allow $\mathcal{A}$ to query FormNym and Open oracles (with the usual constraints) to try to learn information about the unknown user from his pseudonyms. All other actions that $\mathcal{A}$ may want to make the unknown user do (issue credentials, etc.), can be perfectly simulated by $\mathcal{A}$ without knowledge of b. For this, we supply $\mathcal{A}$ with the simulation trapdoor *td*.

Definition 3.3 (Soundness). In the soundness experiment $\text{Exp}^{\text{soundness}}_{\Pi,\mathcal{A}}(\lambda)$, the challenger plays the role of an arbitrary number of honest users. The adversary may internally set up any number of corrupted users.

The experiment begins with handing *pp*, *osk* to the adversary $\mathcal{A}$. We allow $\mathcal{A}$ to make honest users run FormNym and CreateCred, and to interact with honest users running DelegIssue, DelegRcv, or ShowProve. Furthermore, $\mathcal{A}$ can make honest users delegate credentials among themselves.

Eventually, $\mathcal{A}$ outputs a challenge $(pk_{\text{root}}, pk, \phi)$. The experiment runs the protocol ShowVrfy($pp, pk_{\text{root}}, pk, \phi)$ interacting with $\mathcal{A}$. The experiment outputs 1 if ShowVrfy accepts and one of the following is true:

- *The user or root issuer's identity cannot be traced*: $\text{Open}(pp, osk, pk) = \perp$ or $\text{Open}(pp, osk, pk_{\text{root}}) = \perp$.
- *$\mathcal{A}$ was able to impersonate some honest user*: $\text{Open}(pp, osk, pk) = id'$ for some honest user's identity id'.
- *$\mathcal{A}$ was able to show a credential he did not receive*: $\text{Open}(pp, osk, pk_{\text{root}}) = id'$ for some honest user's id' and $\mathcal{A}$ never queried to receive a credential $cred^*$ on a pseudonym pk^* such that: (1) $cred^*$ is rooted at pk_{root}, (2) $cred^*$ has attributes $\vec{A}^* \succeq \phi$, and (3) $cred^*$ is marked delegatable or $\text{Open}(pp, osk, pk) = \text{Open}(pp, osk, pk^*)$.

A DAAC system Π is *sound* if for all ppt adversaries $\mathcal{A}$ there exists a negligible function $negl$ with $\Pr[\text{Exp}^{\text{soundness}}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq negl(\lambda)$ for all $\lambda \in \mathbb{N}$. $\diamond$

The full experiment can be found in Appendix D.

The adversary's win conditions imply that a credential with delegation flag $d = 0$ can only be shown with a pseudonym that opens to the same identity as the pseudonym used when receiving the credential. Note that even if $d = 0$, it is *always* possible for a credential holder A to reveal his credential *and* his user secret usk_A to another user B (who can then show the credential somewhere and gain access). However, users are discouraged from doing this because after revealing usk_A to B, B can perform (malicious) activities, which the opener will then trace back to A. Hence A bears the risk of being made responsible for B's actions. For more details of how we propose applications use the security guarantees of Definition 3.3 to enforce accountability, we refer to Section 3.2.

3.2 How to deploy delegatable attribute-based anonymous credential systems in practice

In the following, we describe an example how an application would utilize DAAC in practice. The system should be set up by a trusted authority (TA). The TA runs $(pp, osk) \leftarrow \text{Setup}(1^\lambda)$, $(usk_{\text{TA}}, id_{\text{TA}}) \leftarrow \text{KeyGen}(pp)$, $(pk_{\text{TA}}, sk_{\text{TA}}) \leftarrow \text{FormNym}(pp, usk_{\text{TA}}, 1^0)$. He then publishes pp and pk_{TA}.

In order to join the system, a user generates $(usk, id) \leftarrow \text{KeyGen}(pp)$. He then approaches the TA to register in the system. For this, he sends one of his pseudonyms pk to the TA and uses some mechanism to authenticate with his real identity (e.g., physically showing a passport). The TA computes the user's id using $\text{Open}(pp, osk, pk)$ and stores id alongside the user's real identity information. Then the TA uses the user's pk to issue a non-delegatable ($d = 0$) "master" credential $cred_{\text{master}}$.

Whenever the user introduces a new pseudonym pk' to some verifier, he first shows the master credential by running $\text{ShowProve}(pp, pk_{\text{TA}}, pk', \phi, sk', usk, cred_{\text{master}})$ with the verifier. This ensures that the user indeed registered with the TA and hence the TA will be able to trace his pseudonyms to his real identity. If at some point the user breaks some rule within the application, the verifier can approach the TA with pk', which the TA can trace to the user's identity by computing $\text{Open}(pp, osk, pk')$. This is because the soundness property (Definition 3.3) ensures that the non-delegatable credential $\mathit{cred}_{\text{master}}$, which was issued to pk, can be successfully shown *only* for pk' where $\text{Open}(\mathit{pp}, \mathit{osk}, \mathit{pk}') = \text{Open}(\mathit{pp}, \mathit{osk}, \mathit{pk})$.

Of course, you may also want to ensure that users cannot be *falsely* accused by verifiers for (malicious) actions they never committed. The application can enforce this by logging relevant actions and making a user certify each log entry by issuing a credential to the verifier whose attributes encode the log entry. This credential (rooted at the user's pseudonym pk') can be used by the verifier to prove to the TA that pk' indeed executed the logged action. Honest users cannot be falsely accused anymore because the soundness property (Definition 3.3) prohibits forging/showing a credential rooted at a pseudonym that traces to an honest user. The user's privacy when issuing the credential to the verifier is still guaranteed because the anonymity property (Definition 3.2) guarantees anonymity not only for the receiver of a credential, but also for the issuer. Note that the TA may still lie about the identity that $\text{Open}(\mathit{pp}, \mathit{osk}, \mathit{pk}')$ outputs. This can be prevented with standard techniques, e.g., by making the TA prove non-interactively that the claimed identity is indeed output by Open.

4 Dynamically malleable signatures with efficient protocols

For our construction of DAAC, we introduce *dynamically malleable signatures* (DMS) as a building block. As explained in the introduction, a DMS is a malleable signature where the set of allowed transformations on the signed message can be incrementally restricted. We first define DMS, then define related protocols that are used in our DAAC construction.

4.1 Definition

A DMS is accompanied by a *malleability key* mk and parameterized with an *index set* I. We describe malleability through a relation $\equiv_I$, which depends on I. Namely, using mk, a message $\vec{m}$ can be changed into a message $\vec{m}'$ iff $\vec{m} \equiv_I \vec{m}'$. We remark that our definitions (syntax and security) for DMS apply to arbitrary equivalence relations $\equiv_I$ and arbitrary index sets I satisfying $I' \subseteq I \Rightarrow \equiv_{I'} \subseteq \equiv_I$, i.e. restricting I restricts the malleability relation. However, in this paper, we are going to use the following concrete relation $\equiv_I$.

Definition 4.1. Let $I \subseteq \{1, \ldots, n\}$ be an index set. We define $\equiv_I$ by

$$(m_1, \ldots, m_n) \equiv_I (m_1', \ldots, m_n') \ \Leftrightarrow \forall i \notin I : \ m_i = m_i' \ . \qquad \diamond$$

This means that malleability of DMS is restricted so that exactly the messages at indices present in I can be modified. A DMS is called *dynamically* malleable because given any signature σ and malleability key mk with index set I, one can efficiently compute σ', mk' with index set $I' \subseteq I$. We now formally define DMS.

Definition 4.2 (Dynamically malleable signatures). A DMS scheme consists of the following (probabilistic) polynomial-time algorithms:

$\text{Setup}(1^\lambda) \to \mathit{pp}$ for security parameter λ outputs public parameters pp. We assume that the message space M can be inferred from pp and that $|\mathit{pp}| \geq \lambda$.

$\text{KeyGen}(\mathit{pp}, 1^n) \to (\mathit{pk}, \mathit{sk})$ for $n \in \mathbb{N}$ outputs a key pair $(\mathit{pk}, \mathit{sk})$.

$\text{Sign}(\mathit{pp}, \mathit{sk}, \vec{m}, I) \to (\sigma, \mathit{mk})$ for a message vector $\vec{m} \in M^n$ and an index set I outputs a signature σ and a malleability key mk.

$\mathrm{Transform}(pp, \vec{m}, \vec{m}', \sigma, mk, I') \to (\sigma', mk')$ on input a signature σ on $\vec{m}$ outputs a signature σ' and a malleability key mk' for $\vec{m}', I'$.

$\mathrm{Vrfy}(pp, pk, \vec{m}, \sigma) = b$ is a deterministic algorithm that outputs a bit.

$\mathrm{VrfyMk}(pp, pk, \vec{m}, \sigma, mk, I) = b$ is a deterministic algorithm that outputs a bit.

A DMS scheme is *correct* if for all $\lambda, n \in \mathbb{N}$, all $pp \in [\mathrm{Setup}(1^\lambda)]$, all $(pk, sk) \in [\mathrm{KeyGen}(pp, 1^n)]$, all $\vec{m} \in M^n$ and index sets $I \subseteq \{1, \dots, n\}$:

- $\Pr[(\sigma, mk) \leftarrow \mathrm{Sign}(pp, sk, \vec{m}, I); \mathrm{Vrfy}(pp, pk, \vec{m}, \sigma) = \mathrm{VrfyMk}(pp, pk, \vec{m}, \sigma, mk, I) = 1] = 1$ (signatures and malleability keys from Sign are accepted by the verification algorithms)
- $\Pr[(\sigma', mk') \leftarrow \mathrm{Transform}(\vec{m}, \vec{m}', \sigma, mk, I'); \mathrm{Vrfy}(pp, pk, \vec{m}', \sigma') = \mathrm{VrfyMk}(pp, pk, \vec{m}', \sigma', mk', I') = 1] = 1$ for all $\vec{m}' \equiv_I \vec{m}$, $I' \subseteq I$, and all (σ, mk) with $\mathrm{VrfyMk}(pp, pk, \vec{m}, \sigma, mk, I) = 1$. (signatures and malleability keys derived from Transform are accepted by the verification algorithms). $\diamond$

Note that our definition implies that any signature/malleability key created with Transform can again be input to Transform to further change the message or weaken the malleability key. Also note that we model both Vrfy and VrfyMk as the former may be more efficient.

We now define security for DMS. We expect (1) derivation privacy: signatures derived with mk are indistinguishable from signatures freshly created with sk and (2) unforgeability: an adversary cannot produce a signature that cannot be legally derived from oracle-queried signatures. For derivation privacy, we demand *perfect* derivation privacy for simplicity.

Definition 4.3 (Perfect derivation privacy). A DMS scheme $\mathcal{S}$ is *perfectly derivation private* if for all $\lambda, n \in \mathbb{N}$, all $pp \in [\mathrm{Setup}(1^\lambda)]$, all pk, σ, mk, all $\vec{m}, \vec{m}' \in M^n$, and all index sets I, I' with $\vec{m}' \equiv_I \vec{m}$, $I' \subseteq I$, and $\mathrm{VrfyMk}(pp, pk, \vec{m}, \sigma, mk, I) = 1$, it holds that

- $\exists sk$ with $(pk, sk) \in [\mathrm{KeyGen}(pp, 1^n)]$
- for all sk such that $(pk, sk) \in [\mathrm{KeyGen}(pp, 1^n)]$, we have that $\mathrm{Transform}(\vec{m}, \vec{m}', \sigma, mk, I') \approx \mathrm{Sign}(pp, sk, \vec{m}', I')$ $\diamond$

The first item (that for each pk accepted by VrfyMk, there exists a corresponding sk) is a somewhat technical requirement. Without this requirement, it may happen that someone receives valid σ, mk from an untrusted source for a public key pk for which there exists no corresponding sk. In this case the premise $(pk, sk) \in [\mathrm{KeyGen}(pp, 1^n)]$ of the second item does not apply and hence any signatures σ' derived from σ, mk would be allowed to be easily traced back to σ.

For the second property, unforgeability, we simply weaken the standard EUF-CMA definition for digital signatures such that signatures that can be legally derived using Transform are not considered forgeries anymore. Note that for simplicity, we only define unforgeability for perfectly derivation private schemes.

Definition 4.4 (Unforgeability). Consider the experiment $\mathtt{SigForge}_{\Pi,\mathcal{A}}(\lambda, n)$ for a DMS scheme Π and an adversary $\mathcal{A}$:

- $pp \leftarrow \mathrm{Setup}(1^\lambda)$, $(pk, sk) \leftarrow \mathrm{KeyGen}(pp, 1^n)$. $\mathcal{A}$ is given pp, pk and oracle access to $\mathrm{Sign}(pp, sk, \cdot, \cdot)$.
- Eventually $\mathcal{A}$ outputs $\vec{m}^*, \sigma^*$. The experiment outputs 1 iff $\mathrm{Vrfy}(pp, pk, \vec{m}^*, \sigma^*) = 1$ and $\mathcal{A}$ never made a query $\mathrm{Sign}(pp, sk, \vec{m}', I')$ where $\vec{m}^* \equiv_{I'} \vec{m}'$.

A perfectly derivation private DMS scheme Π is *unforgeable* if for all polynomials p and all ppt $\mathcal{A}$ there is a negligible function *negl* such that for all $\lambda \in \mathbb{N}$ and $n \leq p(\lambda)$, $\Pr[\texttt{SigForge}_{\Pi,\mathcal{A}}(1^\lambda, 1^n) = 1] \leq negl(\lambda)$. $\diamond$

For perfectly derivation private schemes, the output of Transform is distributed the same as the output of Sign. Hence there is no need to give $\mathcal{A}$ explicit access to a Transform oracle. Furthermore, note that the definition can be fulfilled by schemes where a signature σ can be modified even without a corresponding malleability key mk. Consequently, the distinction between σ and mk is somewhat arbitrary – one may just as well merge mk into σ. However, note that mk is not required as input to Vrfy; hence we keep the distinction for the sake of intuition and potential efficiency gains.

4.2 Deriving a signature on a committed message

For our construction of DAAC, we will require a protocol for deriving a signature on a hidden committed message. The setting for the protocol is the following: The *issuer* holds a signature σ on a message $\vec{m} = (m_1, \dots, m_n)$ and corresponding malleability key mk for index set I. For $i \in I$ and $I^* \subseteq I$ and a message m^*, the *receiver* wants to obtain the output of $\text{Transform}(pp, \vec{m}, (m_1, \dots, m_{i-1}, m^*, m_{i+1}, \dots, m_n), \sigma, mk, I^*)$ without revealing his m^*. For this, the receiver commits to m^*, then both parties engage in a protocol to jointly compute Transform.

Definition 4.5 (Deriving a signature on a committed value). A scheme for deriving a signature on a committed value consists of two ppt algorithms and two interacting algorithms:

$\text{BlindInit}(\sigma, mk, \vec{m}, i) \to (K, k)$ on input a signature σ on $\vec{m}$, an index i, and a corresponding malleability key mk, outputs a key K for the commitment scheme and some secret information k.

$\text{Commit}(K, m^*, r) \to C$ takes as input a key K, a message m^* and some randomness r, and outputs a commitment C.

$\text{BlindIssue}(\sigma, mk, \vec{m}, i, I^*, k, C)$ on input a signature σ on $\vec{m}$, an index set I^*, a malleability key mk, a commitment C, an index i, and the secret k, interacts with BlindRcv.

$\text{BlindRcv}(m^*, i, I^*, K, C, r) \to (\sigma^*, mk^*)$ on input a message m^*, an index i, an index set I^*, a commitment C for key K and its randomness r, interacts with BlindIssue and outputs a signature σ^* and a malleability key mk^*.

The public parameters pp and the public key under which the issuer's signature is valid are considered implicit input to all the algorithms above.

Such a protocol is *correct* if for all $m^* \in M$, all σ, mk valid on $\vec{m}$ with index set I, all $i \in I$, and all $I^* \subseteq I$, the result (σ^*, mk^*) of $\text{BlindRcv}(m^*, i, I^*, K, C, r) \leftrightarrow \text{BlindIssue}(\sigma, mk, \vec{m}, i, I^*, k)$, where $C = \text{Commit}(K, m^*, r)$, is a valid signature (and malleability key) on $(m_1, \dots, m_{i-1}, m^*, m_{i+1}, \dots, m_n)$. $\diamond$

In this scenario, the issuer would use BlindInit to create a commitment key K. He then sends K to the receiver, who uses it to commit to his message m^*. Then both parties engage in the BlindIssue $\leftrightarrow$ BlindRcv protocol, which yields the signature and malleability key for the receiver.

In our credential system construction, we are going to make the receiver prove something about the message m^* that he committed to. For this reason, the commitment process is made explicit in this definition as opposed to hiding it in the implementation details of the BlindRcv $\leftrightarrow$ BlindIssue protocol.

For the security of such a protocol, we require security for the receiver and for the issuer. Security for the receiver requires that (1) the commitment scheme is perfectly hiding, and (2) runs of BlindRcv for two distinct committed messages are perfectly indistinguishable for the issuer. Security for the issuer requires that (1) the distribution of the commitment key K is independent of the issuer's concrete σ, mk and $\vec{m}$, and (2) the receiver only learns a single signature. We detail these requirements formally in Appendix A.

5 Construction of dynamically malleable signatures based on Pointcheval-Sanders signatures

Our construction is an extension of Pointcheval-Sanders signatures [PS16].

The Setup, KeyGen, Vrfy algorithms below are exactly the same as in the original Pointcheval-Sanders signature scheme, as are the signatures produced by Sign. We extend the Sign algorithm to output a malleability key and we add the VrfyMk and Transform algorithms. The main observation for our construction is that a signature $(h, h^{x+\sum y_i m_i})$ on $(m_1, \ldots, m_n)$ can be made malleable at position i by adding h^{y_i} to the malleability key.

Construction 5.1 (DMS scheme).

$\mathrm{Setup}(1^\lambda)$ generates a bilinear group $\mathbb{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e, p)$ of prime order $p \geq 2^\lambda$. It outputs $pp = \mathbb{G}$. The message space is $M = \mathbb{Z}_p$.

$\mathrm{KeyGen}(pp, 1^n)$ chooses random generator $\tilde{g} \leftarrow \mathbb{G}_2$. It then chooses random $x, y_1, \ldots, y_n \leftarrow \mathbb{Z}_p$. The secret key is $sk = (x, y_1, \ldots, y_n)$ and the public key is $pk = (\tilde{g}, \tilde{g}^x, \tilde{g}^{y_1}, \ldots, \tilde{g}^{y_n})$. It outputs (pk, sk).

$\mathrm{Sign}(pp, sk, m_1, \ldots, m_n, I)$ chooses $h \leftarrow \mathbb{G}_1 \setminus \{1\}$ and computes $\sigma = (h, h^{x+\sum_i y_i m_i})$, $mk = (h^{y_i})_{i \in I}$. It outputs (σ, mk).

$\mathrm{Vrfy}(pp, pk, \vec{m}, \sigma)$ parses σ as (h, σ_2), pk as $(\tilde{g}, \tilde{g}^x, \tilde{g}^{y_1}, \ldots, \tilde{g}^{y_n})$, and returns 1 iff $h \neq 1$ and $e(h, \tilde{g}^x \cdot \prod_{i=1}^n (\tilde{g}^{y_i})^{m_i}) = e(\sigma_2, \tilde{g})$.

$\mathrm{VrfyMk}(pp, pk, \vec{m}, \sigma, mk, I)$ checks $\mathrm{Vrfy}(pp, pk, \vec{m}, \sigma) \overset{!}{=} 1$ and outputs 0 if the check does not pass. It parses σ as (h, σ_2) and mk as $(h_i)_{i \in I}$. Then it returns 1 iff $e(h_i, \tilde{g}) = e(h, \tilde{g}^{y_i})$ for all $i \in I$.

$\mathrm{Transform}(\vec{m}, \vec{m}', \sigma, mk, I')$ parses σ as $(h, h^{x+\sum_i y_i m_i})$ and mk as $(h^{y_i})_{i \in I}$ [1]. It aborts if $\mathrm{VrfyMk}(pp, pk, \vec{m}, \sigma, mk, I) \neq 1$ or $I' \not\subseteq I$ or $\vec{m} \not\equiv_I \vec{m}'$. Otherwise it chooses $r \leftarrow \mathbb{Z}_p^*$ and computes

$$\sigma' = (h^r, (h^{x+\sum_i y_i m_i} \cdot \prod_{i \in I} (h^{y_i})^{m'_i - m_i})^r) \text{ and } mk' = ((h^{y_i})^r)_{i \in I'}.$$

It outputs (σ', mk'). $\diamond$

One can easily check that our scheme fulfills the correctness requirements (Definition 4.2). Furthermore, Transform clearly produces signatures with the same distribution as Sign since h^r is uniformly distributed over $\mathbb{G}_1 \setminus \{1\}$ and the second signature component as well as mk are uniquely determined by that choice. Consequently, the scheme is perfectly derivation private (Definition 4.3). The scheme can be proven unforgeable in the generic group model.

[1] VrfyMk guarantees that the signature and the malleability key have this specific form.

Theorem 5.1 (Unforgeability). *Construction 5.1 is unforgeable in the generic group model for type-3 bilinear groups.*

The proof is straight-forward and very similar to other generic group model proofs (e.g., [PS16]). It can be found in Appendix C.

As noted above, for our construction of DAAC, we need an efficient protocol for deriving a signature on a committed value (cf. Definition 4.5). A protocol for *signing* a committed value for the original signature scheme was given in [PS16]. Our protocol is similar, with some changes to account for transforming an existing signature instead of creating new one, and to account for privacy of the issuer (which was not a requirement before). The commitment scheme we use is essentially Pedersen's commitment [Ped91], but we derive the commitment key from the issuer's signature.

Construction 5.2 (Deriving a signature on a committed value).

$\mathrm{BlindInit}(\sigma, mk, \vec{m}, i)$ parses σ as $(h, \sigma_2) \in (\mathbb{G}_1 \setminus \{1\}) \times \mathbb{G}_1$ and $mk = (h^{y_j})_{j \in I}$. It chooses $k \leftarrow \mathbb{Z}_p^*$. It sets $K = ((h^{y_i})^k, h^k)$ and outputs (K, k).

$\mathrm{Commit}(K, m, r)$ parses K as $(h^{y_i k}, h^k) \in \mathbb{G}_1 \times (\mathbb{G}_1 \setminus \{1\})$ and interprets r as an element of $\mathbb{Z}_p$. It outputs the commitment $C = (h^{y_i k})^m \cdot (h^k)^r$.

$\mathrm{BlindRcv}(m^*, i, I^*, K, C, r) \leftrightarrow \mathrm{BlindIssue}(\sigma, mk, \vec{m}, i, I^*, k, C)$ works as follows: BlindIssue parses σ as (h, σ_2) and mk as $(h^{y_i})_{i \in I}$, chooses a random $u \leftarrow \mathbb{Z}_p^*$ and computes $(\sigma_1', \sigma_2') = (h^{ku}, (\sigma_2^k \cdot (h^{y_i})^{-km_i} \cdot C)^u)$. It sends (σ_1', σ_2') together with $mk' = ((h^{y_j})^{ku})_{j \in I^*}$ to the receiver. BlindRcv then unblinds (σ_1', σ_2') as $\sigma^* = (\sigma_1^*, \sigma_2^*) = (\sigma_1', \sigma_2' \cdot (\sigma_1')^{-r})$. BlindRcv outputs (σ^*, mk'). $\diamond$

The proof of security for this construction is straight-forward and can be found in Appendix B.

6 Constructing delegatable attribute-based anonymous credentials from dynamically malleable signatures with efficient protocols

We now construct DAAC from a DMS scheme with efficient protocols. The general construction idea is similar to the generic construction of (attribute-based) credential systems *without* delegation [Lys02], but using DMS instead of standard signatures essentially allows adding the delegation feature. We define the following notation:

Definition 6.1. Let $\mathcal{H}$ be a hash function. For $\vec{A} \in (\mathbb{A} \cup \{\star\})^n$, $d \in \{0,1\}$, usk, and pk_{root}, we define

- $\vec{m}^{(\vec{A}, usk, pk_{\mathrm{root}})} := (m_1, \ldots, m_n, usk, \mathcal{H}(pk_{\mathrm{root}}))$, where $m_i = A_i$ if $m_i \in \mathbb{A}$, and $m_i = 0$ if $A_i = \star$ (where $0 \in M$ denotes some constant).
- $I^{(d, \vec{A})} := \{i \mid A_i = \star \vee (i = n+1 \wedge d = 1)\} \subseteq \{1, \ldots, n+2\}$ $\diamond$

We will also use the special case $\vec{m}^{(\vec{A}, 0, pk_{\mathrm{root}})}$, which is the same as above but with the user secret set to the constant $0 \in M$. In our construction, the predicates CheckPseud, CheckShow, and CheckDeleg from Definition 3.1 can be evaluated in polynomial time and we are going to use them in the algorithm descriptions.

With some details omitted, our construction is going to work as follows: A credential for user usk rooted at pk_{root} with attribute vector $\vec{A}$ and delegation flag d will be a dynamically malleable signature on $\vec{m}^{(\vec{A},usk,pk_{\text{root}})}$ with index set $I^{(d,\vec{A})}$. To delegate a credential, the issuer needs to derive a signature on the receiver's usk without being given usk itself. For this, we use a protocol for deriving a signature on a committed value (cf. Definition 4.5). Showing the credential consists of the user proving possession of usk and a signature with attributes fulfilling some predicate ϕ, such that usk is both within his pseudonym and the signature. The identity of a user with key usk will be $id = f(usk)$ for some one-way function f. Then, following standard techniques, a user's pseudonym is an encryption c of id. For issuing credentials, each pseudonym also contains a signature scheme key $pk_{\mathcal{S}}$ and a signature of knowledge binding the encryption c and $pk_{\mathcal{S}}$ together.

Construction 6.1 (Generic construction of delegatable attribute-based anonymous credentials). Let $\mathcal{S} = (\text{Setup}_{\mathcal{S}}, \text{KeyGen}_{\mathcal{S}}, \text{Sign}_{\mathcal{S}}, \text{Transform}_{\mathcal{S}}, \text{Vrfy}_{\mathcal{S}})$ be a DMS scheme with $(\text{BlindInit}_{\mathcal{S}}, \text{BlindIssue}_{\mathcal{S}}, \text{BlindRcv}_{\mathcal{S}}, \text{Commit}_{\mathcal{S}})$ for deriving a signature on a committed value. Let $\mathcal{E} = (\text{Setup}_{\mathcal{E}}, \text{KeyGen}_{\mathcal{E}}, \text{Encrypt}_{\mathcal{E}}, \text{Decrypt}_{\mathcal{E}})$ be a public-key encryption scheme. Let $\mathcal{OW} = (\text{Setup}_{\mathcal{OW}}, \text{GenFnct}_{\mathcal{OW}})$ be a one-way function scheme. Let $\mathcal{H}$ be a (collision-resistant) hash function (usage hidden within the $\vec{m}^{(\cdots)}$ notation, cf. Definition 6.1).

We require that $\text{Setup}_{\mathcal{S}} = \text{Setup}_{\mathcal{E}} = \text{Setup}_{\mathcal{OW}} =: \text{Setup}$. We denote the (finite) message spaces of $\mathcal{S}$ and $\mathcal{E}$ as $M_{\mathcal{S}}$ and $M_{\mathcal{E}}$, respectively (they may depend on the output of Setup) and require that functions $f \in [\text{GenFnct}_{\mathcal{OW}}(pp)]$ bijectively map between the message spaces for the signature and encryption scheme, i.e. $f : M_{\mathcal{S}} \to M_{\mathcal{E}}$. Furthermore, the hash function must hash to $M_{\mathcal{S}}$, i.e. $\mathcal{H} : \{0,1\}^* \to M_{\mathcal{S}}$. The scheme consists of the following algorithms:

$\text{Setup}(1^\lambda)$ runs $pp_{\mathcal{S},\mathcal{E},\mathcal{OW}} \leftarrow \text{Setup}(1^\lambda)$, $(pk_{\mathcal{E}}, sk_{\mathcal{E}}) \leftarrow \text{KeyGen}_{\mathcal{E}}(pp_{\mathcal{S},\mathcal{E},\mathcal{OW}})$, and also it chooses $f \leftarrow \text{GenFnct}_{\mathcal{OW}}(pp_{\mathcal{S},\mathcal{E},\mathcal{OW}})$. It outputs $pp = (pp_{\mathcal{S},\mathcal{E},\mathcal{OW}}, pk_{\mathcal{E}}, f)$ and the opening key $osk = sk_{\mathcal{E}}$. The attribute universe $\mathbb{A}$ is $M_{\mathcal{S}}$.

$\text{KeyGen}(pp)$ chooses $usk \leftarrow M_{\mathcal{S}}$ and sets $id = f(usk)$. It returns (usk, id).

$\text{FormNym}(pp, usk, 1^n)$ generates keys $(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \text{KeyGen}_{\mathcal{S}}(pp_{\mathcal{S},\mathcal{E},\mathcal{OW}}, 1^{n+2})$. It encrypts usk as $c = \text{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(usk); r')$ using fresh randomness r'. It then creates a signature of knowledge on $pk_{\mathcal{S}}$ and c proving that it can open c: $\gamma = \mathit{NIZK}[(usk, r'); c = \text{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(usk); r')](pk_{\mathcal{S}}, c)$. The pseudonym is $pk = (pk_{\mathcal{S}}, c, \gamma)$ and the secret is $sk = (sk_{\mathcal{S}}, r', usk)$. It outputs (pk, sk).

$\text{Open}(pp, osk, pk)$ parses pk as $(pk_{\mathcal{S}}, c, \gamma)$, checks γ (outputs $\perp$ and aborts if the check fails), then it runs and outputs $\text{Decrypt}_{\mathcal{E}}(sk_{\mathcal{E}}, c)$.

$\text{CreateCred}(pp, pk, sk)$ runs $(\sigma, mk) \leftarrow \text{Sign}_{\mathcal{S}}(sk_{\mathcal{S}}, \vec{m}^{((\star,\dots,\star),usk,pk)}, I^{(1,(\star,\dots,\star))})$. It outputs $cred = (\sigma, mk, (\star, \dots, \star), d = 1)$

$\text{DelegIssue}(pp, pk_{\text{root}}, usk, cred, \vec{A}^*, d^*, pk^*) \leftrightarrow \text{DelegRcv}(pp, pk_{\text{root}}, \vec{A}^*, d^*, pk^*, sk^*, usk^*)$ works as follows:

1. The issuer checks $\text{CheckDeleg}(pp, pk_{\text{root}}, usk, cred, \vec{A}^*) \overset{!}{=} 1$ and parses $cred$ as $(\sigma, mk, \vec{A}, d)$.
2. The receiver checks $\text{CheckPseud}(pp, pk^*, sk^*, usk^*) \overset{!}{=} 1$.
3. The issuer prepares an anonymized delegatable credential by running $(\sigma', mk') \leftarrow \text{Transform}(pp, \vec{m}^{(\vec{A},usk,pk_{\text{root}})}, \vec{m}^{(\vec{A}^*,0,pk_{\text{root}})}, \sigma, mk, I^{(1,\vec{A}^*)})$.

4. If $d^* = 1$, the issuer simply hands (σ', mk') to the receiver. Then the receiver changes the signature to his user secret by running $(\sigma^*, mk^*) \leftarrow \mathrm{Transform}(pp, \vec{m}^{(\vec{A}^*,0,pk_{\mathrm{root}})}, \vec{m}^{(\vec{A}^*,usk^*,pk_{\mathrm{root}})}, \sigma', mk', I^{(d^*,\vec{A}^*)})$
5. If $d^* = 0$, we write $pk_{\mathrm{root}} = (pk_{\mathcal{S},\mathrm{root}}, c_{\mathrm{root}}, \gamma_{\mathrm{root}})$. Then
   (a) the issuer runs $(K, k) \leftarrow \mathrm{BlindInit}_{\mathcal{S}}(\sigma', mk', \vec{m}^{(\vec{A}^*,0,pk_{\mathrm{root}})}, n+1)$, sends K to the receiver.
   (b) the receiver computes $C \leftarrow \mathrm{Commit}_{\mathcal{S}}(K, usk^*, r)$ for some random r, sends C to the issuer, and then runs a zero knowledge argument of knowledge with the issuer, proving he can open the commitment and his pseudonym $pk^* = (pk^*_{\mathcal{S}}, c^*, \gamma^*)$ to his user secret (using r' and usk^* from $sk^* = (sk^*_{\mathcal{S}}, r', usk^*)$):
   $$\begin{aligned} ZKAK[(usk^*, r, r'); C &= \mathrm{Commit}_{\mathcal{S}}(K, usk^*, r) \\ &\wedge c^* = \mathrm{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(usk^*); r')] \end{aligned}$$
   (c) if the $ZKAK$ protocol accepts, then the issuer runs the protocol $\mathrm{BlindIssue}_{\mathcal{S}}(\sigma', mk', \vec{m}^{(\vec{A}^*,0,pk_{\mathrm{root}})}, n{+}1, I^{(d^*,\vec{A}^*)}, k, C)$, while the receiver runs $(\sigma^*, mk^*) \leftarrow \mathrm{BlindRcv}_{\mathcal{S}}(usk^*, n+1, I^{(d^*,\vec{A}^*)}, K, C, r))$.
6. The receiver checks $\mathrm{VrfyMk}_{\mathcal{S}}(pk_{\mathcal{S},\mathrm{root}}, \vec{m}^{(\vec{A}^*,usk^*,pk_{\mathrm{root}})}, \sigma^*, mk^*, I^{(d^*,\vec{A}^*)}) \overset{!}{=} 1$. If the check fails, it outputs $\perp$, otherwise it outputs $cred^* := (\sigma^*, mk^*, \vec{A}^*, d^*)$.

$\mathrm{ShowVrfy}(pp, pk_{\mathrm{root}}, pk, \phi) \leftrightarrow \mathrm{ShowProve}(pp, pk_{\mathrm{root}}, pk, \phi, sk, usk, cred)$ is as follows: We write $pk_{\mathrm{root}} = (pk_{\mathcal{S},\mathrm{root}}, \cdot, \cdot)$, $pk = (\cdot, c, \cdot)$ and $sk = (sk_{\mathcal{S}}, r', usk)$.

1. The prover checks $\mathrm{CheckShow}(pp, pk_{\mathrm{root}}, pk, \phi, sk, usk, cred) \overset{!}{=} 1$.
2. The prover parses $cred = (\sigma, mk, \vec{A}, d)$ and computes some $\vec{A}' \in \mathbb{A}^n$ with $\vec{A}' \preceq \vec{A}$ and $\phi(\vec{A}') = 1$. [2]
3. The prover then runs $(\sigma', mk') \leftarrow \mathrm{Transform}(pp, \vec{m}^{(\vec{A},usk,pk_{\mathrm{root}})}, \vec{m}^{(\vec{A}',usk,pk_{\mathrm{root}})}, \sigma, mk, I)$ with $I = \emptyset$.
4. The prover runs the following black-box zero-knowledge argument of knowledge protocol with the verifier:
$$\begin{aligned} ZKAK[(usk, r', \sigma', \vec{A}'); &\mathrm{Vrfy}_{\mathcal{S}}(pk_{\mathcal{S},\mathrm{root}}, \vec{m}^{(\vec{A}',usk,pk_{\mathcal{S},\mathrm{root}})}, \sigma') = 1 \\ &\wedge \phi(\vec{A}') = 1 \wedge c = \mathrm{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(usk); r')] \end{aligned}$$

The checker predicates required by Definition 3.1 are as follows. We denote them as algorithms because in our case the predicates are polynomial-time computable.

$\mathrm{CheckPseud}(pp, pk, sk, usk)$ outputs 1 if and only if $pk = (pk_{\mathcal{S}}, c, \gamma)$ and $sk = (sk_{\mathcal{S}}, r', usk)$ such that $c = \mathrm{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(usk); r')$ and γ is valid signature of knowledge for $NIZK[(usk, r'); c = \mathrm{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(usk); r')](pk_{\mathcal{S}}, c)$.

$\mathrm{CheckShow}(pp, pk_{\mathrm{root}}, pk, \phi, sk, usk, cred)$ outputs 1 iff $\mathrm{CheckPseud}(pp, pk, sk, usk) = 1$, and $pk_{\mathrm{root}} = (pk_{\mathcal{S},\mathrm{root}}, c_{\mathrm{root}}, \gamma_{\mathrm{root}})$, $cred = (\sigma, mk, \vec{A}, d)$ such that $\vec{A} \succeq \phi$, $\mathrm{VrfyMk}_{\mathcal{S}}(pk_{\mathcal{S},\mathrm{root}}, \vec{m}^{(\vec{A},usk,pk_{\mathrm{root}})}, \sigma, mk, I^{(d,\vec{A})}) = 1$, and γ_{root} is a valid signature of knowledge $NIZK[(usk_{\mathrm{root}}, r_{\mathrm{root}}); c_{\mathrm{root}} = \mathrm{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(usk_{\mathrm{root}}); r_{\mathrm{root}})](pk_{\mathcal{S},\mathrm{root}}, c_{\mathrm{root}})$

[2] we assume that the set of valid ϕ allows an efficient computation of such an $\vec{A}'$, e.g,. ϕ is given in disjunctive normal form, or n is upper-bounded by some constant n_{max}.

CheckDeleg$(pp, pk_{\text{root}}, usk, cred, \vec{A}^*)$ outputs 1 if and only if $pk_{\text{root}} = (pk_{\mathcal{S},\text{root}}, c_{\text{root}}, \gamma_{\text{root}})$, and $cred = (\sigma, mk, \vec{A}, d)$ such that γ_{root} is a valid signature of knowledge with respect to $NIZK[(usk_{\text{root}}, r_{\text{root}}); c_{\text{root}} = \text{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(usk_{\text{root}}); r_{\text{root}})](pk_{\mathcal{S},\text{root}}, c_{\text{root}})$, and $d = 1$, $\vec{A} \succeq \vec{A}^*$, and $\text{VrfyMk}_{\mathcal{S}}(pk_{\mathcal{S},\text{root}}, \vec{m}^{(\vec{A},usk,pk_{\text{root}})}, \sigma, mk, I^{(d,\vec{A})}) = 1$ $\diamond$

Note that parameters for the argument of knowledge, signature of knowledge, and the hash function $\mathcal{H}$ also need to be part of the public parameters pp, but we abstract from the details here.

One can instantiate this construction in a type-3 bilinear group setting, i.e. Setup generates a bilinear group $\mathbb{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p)$ of prime order p. You can use the DMS scheme and its protocol from Section 5 for $\mathcal{S}$, and Cramer-Shoup encryption [CS98] in $\mathbb{G}_1$ for $\mathcal{E}$. Then $M_{\mathcal{S}} = \mathbb{Z}_p$ and $M_{\mathcal{E}} = \mathbb{G}_1$, so the bijective one-way function $\mathcal{OW}$ can simply be $x \mapsto g^x$. In this setting, the statements in the zero-knowledge arguments of knowledge fall into the *prove knowledge of exponents* category. Hence they can be easily constructed from Schnorr-like Σ protocols together with Damgård's technique [Dam00] (using Pedersen's commitment) to make it concurrent black-box zero-knowledge. To enable arbitrary Boolean formulas for policies ϕ, one can combine the Σ protocol with proofs of partial knowledge [CDS94]. For the signatures of knowledge, you can use Schnorr-like protocols with the Fiat-Shamir heuristic.

Theorem 6.1 (Security of the generic construction)**.** *If $\mathcal{E}$ is correct and CCA-secure, $\mathcal{S}$ is correct (Definition 4.2), unforgeable (Definition 4.4) and perfectly derivation private (Definition 4.3) with secure protocol for deriving a signature on a committed value (Definitions 4.5,A.1,A.2), $\mathcal{OW}$ is a secure one-way function scheme, and $\mathcal{H}$ is a collision-resistant hash function, then Construction 6.1 is* correct *(Definition 3.1),* anonymous *(Definition 3.2), and* sound *(Definition 3.3).*

A sketch for the proof can be found in Appendix E.

7 Conclusion and future work

We introduced the notion of DAAC and gave a generic construction from DMS. Our generic construction can be instantiated using the DMS scheme from Section 5, which results in a very efficient and expressive DAAC. Besides the relatively short signatures, this is also due to the fact that all required protocols fall into the *zero knowledge proof of exponents* category, for which there are many efficient protocols.

While other existing delegatable credential schemes (e.g., [BCC+09]) allow extracting the complete delegation history of a credential (i.e. a complete list of who delegated to whom), our scheme only allows the opener to trace the first and the last user of a delegation chain. While we assume that this is sufficient for many applications, some may prefer to be able to check the complete delegation history of a credential. We leave it for future work to extend our approach.

In our construction, the *covers* relation prevents users from delegating certain attributes. While this delegation policy seems to be sufficient for many typical applications, the following question remains open: *How can (efficient) attribute-based credentials with other/more general delegation relations be constructed?*

On the topic of DMS, our concrete construction showed that such signature schemes exist, but our construction is proven secure only under an untested algebraic assumption. Hence we identify the following research question for future work: *Can DMS be constructed from standard assumptions?* A good starting point for answering this question may be a Naor-like transformation on spatial encryption (e.g., [CW14, AL11]), where keys represent linear subspaces of a vector space.

References

[ABC$^+$15] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. *Journal of Cryptology*, 28(2), 2015.

[AL11] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In *PKC 2011*, LNCS. Springer, 2011.

[AN11] Tolga Acar and Lan Nguyen. Revocation for delegatable anonymous credentials. In *PKC 2011*, LNCS. Springer, 2011.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. Cryptology ePrint Archive, Report 2005/015, 2005. `http://eprint.iacr.org/2005/015`.

[BCC$^+$09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO 2009*, LNCS. Springer, 2009.

[BFKW09] Dan Boneh, David Mandell Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *PKC 2009*, LNCS. Springer, 2009.

[CDD17] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical uc-secure delegatable credentials with attributes and their application to blockchain. In *CCS*, pages 683–699. ACM, 2017.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO 1994*, LNCS. Springer, 1994.

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

[CKLM14] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: New definitions and delegatable anonymous credentials. In *CSF 2014*. IEEE, 2014.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO 2006*, LNCS. Springer, 2006.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO 1998*, LNCS. Springer, 1998.

[CW14] Jie Chen and Hoeteck Wee. Doubly spatial encryption from DBDH. *Theoretical Computer Science*, 543, 2014.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT 2000*, LNCS. Springer, 2000.

[Fre12] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *PKC 2012*, LNCS. Springer, 2012.

[Fuc10] Georg Fuchsbauer. Commuting signatures and verifiable encryption and an application to non-interactively delegatable credentials. *IACR Cryptology ePrint Archive*, 2010.

[Lys02] Anna Lysyanskaya. *Signature schemes and applications to cryptographic protocol design.* PhD thesis, Massachusetts Institute of Technology, 2002.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO 1991*, LNCS. Springer, 1991.

[PS16] David Pointcheval and Olivier Sanders. Short randomizable signatures. In *CT-RSA 2016*, LNCS. Springer, 2016.

A Defining security for deriving a signature on a committed message

For the receiver, we require that the commitment hides the message m^* and that the BlindRcv protocol (or its success/failure to output a valid signature) does not reveal anything about m^* either.

Definition A.1 (Security for the receiver). A scheme for deriving a signature on a committed value is *secure for the receiver* if

- *"the commitment scheme is perfectly hiding"*: For all $m_0^*, m_1^* \in M$, $\mathrm{Commit}(K, m_0^*, r)$ is distributed the same as $\mathrm{Commit}(K, m_1^*, r')$ over the random choice of r, r'.
- "BlindRcv *does not reveal the message"*: for all $\vec{m}$ and any two messages $m_0^*, m_1^* \in M$, $i \in \mathbb{N}$, all K, and all (unrestricted) adversaries $\mathcal{A}$:

$$\begin{aligned}&(\mathrm{output}_{\mathcal{A}}[\mathcal{A}(C_0) \leftrightarrow \mathrm{BlindRcv}(m_0^*, i, I^*, K, C_0, r)], \chi_0)\\ \approx\ &(\mathrm{output}_{\mathcal{A}}[\mathcal{A}(C_1) \leftrightarrow \mathrm{BlindRcv}(m_1^*, i, I^*, K, C_1, r)], \chi_1)\end{aligned}$$

  where r is chosen uniformly at random, $C_j = \mathrm{Commit}(K, m_j^*, r)$ and χ_j is an indicator variable with $\chi_j = 1$ if and only if $\mathrm{Vrfy}(pp, pk, (m_1, \ldots, m_{i-1}, m_j^*, m_{i+1}, \ldots, m_n), \sigma^*) = \mathrm{VrfyMk}(pp, pk, (m_1, \ldots, m_{i-1}, m_j^*, m_{i+1}, \ldots, m_n), \sigma^*, mk^*, I^*) = 1$ for the output (σ^*, mk^*) of BlindRcv. $\diamond$

For the security for the *issuer* we require that (1) the commitment key does not reveal anything about the signature that the issuer holds, and (2) the receiver does not learn more than a signature and its malleability key from the BlindIssue protocol.

Definition A.2 (Security for the issuer). A scheme for deriving a signature on a committed value is *secure for the issuer* if the following hold:

- *"The commitment key is independent of σ, mk, and $\vec{m}$"*: for all $pk, \sigma, mk, mk', \vec{m}, \vec{m}', I, I', i$ such that $\mathrm{VrfyMk}(pp, pk, \vec{m}, \sigma, mk, I) = \mathrm{VrfyMk}(pp, pk, \vec{m}', \sigma', mk', I') = 1$ and $i \in I \cap I'$, it holds that for all K^*,

$$\begin{aligned}&\Pr[(K, k) \leftarrow \mathrm{BlindInit}(\sigma, mk, \vec{m}, i); K = K^*]\\ =&\Pr[(K, k) \leftarrow \mathrm{BlindInit}(\sigma', mk', \vec{m}', i); K = K^*]\end{aligned}$$

- *"The receiver does not learn too much"*: There is a ppt simulator $\mathfrak{S}$ such that for all (unbounded) algorithms $\mathcal{A}$, for all $pk, \sigma, mk, \sigma^*, mk^*, \vec{m}, \vec{m}^*, m^*, i, I, I^*, K, k, C, r$ where $\text{VrfyMk}(pp, pk, \vec{m}, \sigma, mk, I) = 1$, $i \in I$, $I^* \subseteq I$, and $(K, k) \in [\text{BlindInit}(\sigma, mk, \vec{m}, i)]$, $C = \text{Commit}(K, m^*, r)$, and $\vec{m}^* = (m_1, \dots, m_{i-1}, m^*, m_{i+1}, \dots, m_n)$, $\text{VrfyMk}(pp, pk, \vec{m}^*, \sigma^*, mk^*, I^*) = 1$, the following distributions are identical:

$$\begin{aligned} &\text{output}_{\mathcal{A}}[\mathcal{A} \leftrightarrow \text{BlindIssue}(\sigma, mk, \vec{m}, i, I^*, k, C)] \\ \approx\ &\text{output}_{\mathcal{A}}[\mathcal{A} \leftrightarrow \mathfrak{S}_{\text{BlindIssue}}(m^*, i, I^*, K, C, r, \vec{m}^*, \sigma^*, mk^*)] \end{aligned} \qquad \diamond$$

In this definition, we give the simulator $\mathfrak{S}_{\text{BlindIssue}}$ the input of BlindRcv (namely m^*, i, I^*, K, C, r) plus what the receiver is allowed to additionally learn from the protocol. That is the target message $\vec{m}^* = (m_1, \dots, m_{i-1}, m^*, m_{i+1}, \dots, m_n)$, a signature σ^* on $\vec{m}^*$, and a corresponding malleability key mk^*.

B Security proofs for the protocol for deriving a signature on a committed value

Theorem B.1 (Correctness)**.** *Construction 5.2 is* correct*.*

Proof. If both parties follow the protocol, then for the (σ_1', σ_2') computed by BlindIssue, it holds that

$$(\sigma_1', \sigma_2') = (h^{ku}, (h^{ku})^{x + \sum_{j \neq i} y_j m_j + y_i m^*} \cdot h^{kru}) \ ,$$

i.e. it is a correct signature on $(m_1, \dots, m_{i-1}, m^*, m_{i+1}, \dots, m_n)$ except for the h^{kru} term. Hence BlindRcv unblinds the signature correctly by multiplying it with $(h^{ku})^{-r}$. It is easy to see that mk' has the right form. $\square$

Theorem B.2. *Construction 5.2 has security for the issuer and security for the receiver.*

Proof. **Security for the receiver**:

- The commitment scheme is obviously perfectly hiding: $\text{Commit}(K, m, r)$ outputs a uniformly element $C \in \mathbb{G}_1$ over the random choice of r.
- BlindRcv does not send any messages, hence obviously the output of any algorithm $\mathcal{A}$ interacting with BlindRcv is independent of the input to BlindRcv. Also, for every (σ_1', σ_2') that $\mathcal{A}$ may send, there exists exactly one r such that unblinding results in a valid signature. Consequently, since $\mathcal{A}$'s view (consisting only of the commitment C) is independent of r and m^*, failure and success of BlindRcv are also independent of m^*.

Security for the issuer: First, the commitment key K output by BlindInit is (g^{y_i}, g) for some uniformly random $g \leftarrow \mathbb{G} \setminus \{1\}$, hence independent of $\sigma, mk, \vec{m}$.

For the other property, define $\mathfrak{S}_{\text{BlindIssue}}(m^*, i, I^*, K, C, r, \vec{m}^*, \sigma^*, mk^*)$ interacting with $\mathcal{A}$ as follows: $\mathfrak{S}_{\text{BlindIssue}}$ parses K as $(h^{y_i k}, h^k) \in \mathbb{G}_1 \times (\mathbb{G}_1 \setminus \{1\})$, $\vec{m}^* = (m_1, \dots, m_{i-1}, m^*, m_{i+1}, \dots, m_n)$, and parses σ^* as $(h^*, (h^*)^{\sum_{j \neq i} y_j m_j + y_i m^*})$, and as $mk^* = ((h^*)^{y_j})_{j \in I^*}$. It chooses $u \leftarrow \mathbb{Z}_p^*$ and then sets

$$\sigma' = (\sigma_1', \sigma_2') = ((h^*)^u, ((h^*)^u)^{\sum_{j \neq i} y_j m_j + y_i m^*} \cdot (h^*)^{ur}) \ ,$$

and $mk' = ((h^*)^{y_j u})_{j \in I^*}$. $\mathfrak{S}_{\text{BlindIssue}}$ then sends σ', mk' to $\mathcal{A}$.

Note that if $C = h^{y_i k m^*} \cdot h^{kr}$, then BlindIssue on input $(\sigma, mk, \vec{m}, i, I^*, k, C)$ sends a blinded signature in the form $(h', (h')^{\sum_{j\neq i} y_j m_j + y_i m^*} \cdot (h')^r)$ and corresponding $mk' = ((h')^{y_j})_{j\in I^*}$ where h' is uniformly random in $\mathbb{G}_1 \setminus \{1\}$ (h' corresponds to h^{ku} in Construction 5.2). The same distribution is created by $\mathfrak{S}_{\text{BlindIssue}}$ (for which h' corresponds to $(h^*)^u$). Hence the one message sent by BlindIssue and $\mathfrak{S}_{\text{BlindIssue}}$ follows the same distribution for both, which immediately implies the second property of security for the receiver. □

C Generic group model proof of the dynamically malleable signature scheme

We prove that our DMS scheme (Construction 5.1) is secure in the generic group model, i.e. no generic ppt algorithm has non-negligible chance to produce a forgery.

Theorem 5.1. Let $\mathcal{A}$ be a generic ppt adversary against the unforgeability game. $\mathcal{A}$ may query oracles for group- and pairing operations and for signatures. Using the usual generic group model proof outline, we respond to $\mathcal{A}$'s group oracle queries using random encodings for polynomials over $\mathbb{Z}_p$ in the variables $x, y_1, \dots, y_n, r_1, \dots, r_q$, where $x, y_1, \dots, y_n$ correspond to the exponents of the secret key and $r_1, \dots, r_q$ correspond to the randomization values used for signature queries. Let q be an upper bound for the number of signature queries that $\mathcal{A}$ makes. Without loss of generality, we assume that $\mathcal{A}$ queries the group oracles for the candidate forgery it outputs. We prove that the probability that $\mathcal{A}$ outputs a forgery is negligible in the security parameter λ.

We now give $\mathcal{A}$ random encodings of $x, y_1, \dots, y_n$ from the second group as input (this corresponds to pk). The kth signature query for $(\vec{m}^{(k)}, I^{(k)})$ is answered with encodings of

$$\sigma_k = \left(r_k,\ r_k \cdot \left(x + \sum_{i=1}^n y_i m_i^{(k)}\right)\right) \text{ and } mk_k = (r_k y_j)_{j\in I^{(k)}}$$

in the first group.

Eventually $\mathcal{A}$ outputs $\vec{m}^* \in \mathbb{Z}_p^n$ and encodings corresponding to $\sigma^* = (r^*, z^*) \in \mathbb{Z}_p[x, y_1, \dots, y_n, r_1, \dots, r_q]^2$. Since the only values that $\mathcal{A}$ gets from the signing oracle in the first group are signatures and their malleability keys, r^* and z^* as polynomials are linear combinations of the terms r_k, $r_k \cdot (x + \sum_i y_i m_i^{(k)})$, and $r_k y_j$ for $k \in \{1, \dots, q\}, j \in I^{(k)}$. We write

$$r^* = \sum_{k=1}^q (u_k \cdot r_k + v_k \cdot r_k(x + \sum_{i=1}^n y_i m_i^{(k)})) + \sum_{k=1}^q \sum_{j\in I^{(k)}} s_{k,j} r_k y_j$$

and

$$z^* = \sum_{k=1}^q (u'_k \cdot r_k + v'_k \cdot r_k(x + \sum_{i=1}^n y_i m_i^{(k)})) + \sum_{k=1}^q \sum_{j\in I^{(k)}} s'_{k,j} r_k y_j$$

for suitable coefficients $u_k, u'_k, v_k, v'_k, s_{k,j}, s'_{k,j} \in \mathbb{Z}_p$ (for $k \in \{1, \dots, q\}, j \in \{1, \dots, n\}$).

Corresponding to the (generic) Vrfy algorithm, $\sigma^* = (r^*, z^*)$ is a valid forgery on $\vec{m}^*$ if and only if

$$r^* \neq 0, \tag{1}$$

$$z^* = r^* \left(x + \sum_{i=1}^n y_i m_i^*\right), \tag{2}$$

and

$$\forall k \in \{1 \ldots, q\}\ \exists i \in \{1, \ldots, n\} :\ i \notin I^{(k)} \wedge m_i^{(k)} \neq m_i^* \tag{3}$$

(if (3) does not hold, then $\mathcal{A}$ has queried a signature with a malleability key that enables him to legally derive a signature on m^*).

For contradiction, assume that σ^* is a valid forgery. Then

$$\begin{aligned}
&z^* = r^*(x + \sum_i y_i m_i^*) \\
&\Rightarrow \sum_k \left(u'_k \cdot r_k + v'_k \cdot r_k \left(x + \sum_i y_i m_i^{(k)} \right) \right) + \sum_k \sum_{j \in I^{(k)}} s'_{k,j} r_k y_j \\
&\quad = \left(\sum_k \left(u_k \cdot r_k + v_k \cdot r_k \left(x + \sum_i y_i m_i^{(k)} \right) \right) + \sum_k \sum_{j \in I^{(k)}} s_{k,j} r_k y_j \right) \cdot \left(x + \sum_i y_i m_i^* \right)
\end{aligned}$$

Note that on the left hand side, there are no terms $r_k x^2$, hence $v_k = 0$, and no terms $r_k y_j x$, hence $s_{k,j} = 0$. There is no term r_k on the right hand side, hence $u'_k = 0$. Plugging in the zeros, we get

$$\underbrace{\sum_k \left(v'_k \cdot r_k (x + \sum_i y_i m_i^{(k)}) + \sum_{j \in I^{(k)}} s'_{k,j} r_k y_j \right)}_{=z^*} = \underbrace{\sum_k \left(u_k \cdot r_k \left(x + \sum_i y_i m_i^* \right) \right)}_{=r^*(x + \sum_i y_i m_i^*)}$$

Note that $v'_k = u_k$ for all k as otherwise the coefficients for $r_k x$ do not match. From (1) and (2), we know that $z^* \neq 0$ and hence for at least one k we have $u_k = v'_k \neq 0$. For that k, we get from (3) that there is an i where $m_i^{(k)} \neq m_i^*$ and the term $r_k y_i$ does not appear in the sum $\sum_{j \in I^{(k)}} s'_{k,j} r_k y_j$ on the left hand side. Hence for that k and i, the coefficients of the term $r_k y_i$ on the left hand side (namely $v'_k \cdot m_i^{(k)}$) and the right hand side (namely $u_k \cdot m_i^*$) differ, which contradicts our assumption that σ^* is a valid signature.

Finally, a standard argument (e.g., [BBG05]) shows that with overwhelming probability, our simulated oracles behave exactly like the proper $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ group oracles (in which case $\mathcal{A}$ *cannot* produce a valid signature). Applying the Schwartz-Zippel lemma, we get that the probability for $\mathcal{A}$ to succeed in the forging experiment is at most $(q^*)^2/(p-1)$, where q^* is an upper bound for the number of group element encodings that $\mathcal{A}$ sees. □

D Formal security model for delegatable attribute-based anonymous credentials

Definition D.1 (Correctness). A DAAC system is *correct* if for all $\lambda, n, n' \in \mathbb{N}$ and all $(pp, osk) \in [\text{Setup}(1^\lambda)]$

- "Honestly generated pseudonyms are valid": For all $(usk, id) \in [\text{KeyGen}(pp)]$, $(pk, sk) \in [\text{FormNym}(pp, usk, 1^n)]$:

$$\text{CheckPseud}(pp, pk, sk, usk) = 1$$

- "A user's pseudonyms open to his identity": For all $(usk, id) \in [\mathrm{KeyGen}(pp)]$ and all (pk, sk) with $\mathrm{CheckPseud}(pp, pk, sk, usk) = 1$:

$$\mathrm{Open}(pp, osk, pk) = id$$

- "Root credentials are universal": For all $(usk, id) \in [\mathrm{KeyGen}(pp)]$, $(pk_{\mathrm{root}}, sk_{\mathrm{root}}) \in [\mathrm{FormNym}(pp, usk, 1^n)]$, $(pk, sk) \in [\mathrm{FormNym}(pp, usk, 1^{n'})]$, all $\vec{A}^* \in (\mathbb{A} \cup \{\star\})^n$ and all satisfiable $\phi : \mathbb{A}^n \to \{0,1\}$, it needs to hold that if $cred \in [\mathrm{CreateCred}(pp, pk_{\mathrm{root}}, sk_{\mathrm{root}})]$, then

$$\begin{aligned}&\mathrm{CheckShow}(pp, pk_{\mathrm{root}}, pk, \phi, sk, usk, cred) = 1\\ \text{and } &\mathrm{CheckDeleg}(pp, pk_{\mathrm{root}}, usk, cred, \vec{A}^*) = 1\end{aligned}$$

- "The show protocol succeeds": For all $pk_{\mathrm{root}}, pk, \phi, sk, usk, cred$,

$$\begin{aligned}&\mathrm{CheckShow}(pp, pk_{\mathrm{root}}, pk, \phi, sk, usk, cred) = 1 \land \mathrm{CheckPseud}(pp, pk, sk, usk) = 1\\ \Rightarrow &\Pr[\mathrm{output}_{\mathrm{ShowVrfy}}[\mathrm{ShowVrfy}(pp, pk_{\mathrm{root}}, pk, \phi)\\ &\qquad \leftrightarrow \mathrm{ShowProve}(pp, pk_{\mathrm{root}}, pk, \phi, sk, usk, cred)] = 1] = 1\end{aligned}$$

- "The delegate protocol succeeds": For all $pk_{\mathrm{root}}, usk, cred, \vec{A}^*, d^*, pk^*, sk^*, usk^*$:

$$\begin{aligned}&\mathrm{CheckDeleg}(pp, pk_{\mathrm{root}}, usk, cred, \vec{A}^*) = 1\\ \land &\mathrm{CheckPseud}(pp, pk^*, sk^*, usk^*) = 1\\ \Rightarrow &\Pr[\mathrm{output}_{\mathrm{DelegRcv}}[\mathrm{DelegRcv}(pp, pk_{\mathrm{root}}, \vec{A}^*, d^*, pk^*, sk^*, usk^*)\\ &\qquad \leftrightarrow \mathrm{DelegIssue}(pp, pk_{\mathrm{root}}, usk, cred, \vec{A}^*, d^*, pk^*)] \neq \perp] = 1\end{aligned}$$

- "Any output by DelegRcv (except $\perp$) is a valid credential with the expected delegation and showing properties": For all $pk_{\mathrm{root}}, \vec{A}^*, \vec{A}', d^*, pk^*, sk^*, usk^*, pk', sk', \phi$, all algorithms $\mathcal{A}$ and $cred^* \in [\mathrm{output}_{\mathrm{DelegRcv}}[\mathrm{DelegRcv}(pp, pk_{\mathrm{root}}, \vec{A}^*, d^*, pk^*, sk^*, usk^*) \leftrightarrow \mathcal{A}]]$, the following must hold:

1. $$\begin{aligned}&cred^* \neq \perp\\ \land &\mathrm{CheckPseud}(pp, pk', sk', usk^*) = 1\\ \land &\vec{A}^* \succeq \phi\\ \Rightarrow &\mathrm{CheckShow}(pp, pk_{\mathrm{root}}, pk', \phi, sk', usk^*, cred^*) = 1\end{aligned}$$

2. $$\begin{aligned}&cred^* \neq \perp\\ \land &\vec{A}^* \succeq \vec{A}' \land d^* = 1\\ \Rightarrow &\mathrm{CheckDeleg}(pp, pk_{\mathrm{root}}, usk^*, cred^*, \vec{A}') = 1\end{aligned}$$

◇

Definition D.2 (Anonymity). A DAAC system Π has *anonymity* if there is an ppt algorithm $(pp, osk, td) \leftarrow \mathfrak{S}^{pp}(1^\lambda)$ whose output is such that (pp, osk) is distributed exactly like $\mathrm{Setup}(1^\lambda)$. Furthermore, there are ppt simulators $\mathfrak{S}_{\mathrm{ShowProve}}, \mathfrak{S}_{\mathrm{DelegIssue}}, \mathfrak{S}_{\mathrm{DelegRcv}}$ such that for all $\lambda \in \mathbb{N}$ and $(pp, osk, td) \in [\mathfrak{S}^{pp}(1^\lambda)]$ and all (unrestricted) $\mathcal{A}$:

1. *"Anonymity when showing a credential"*: For all $pk_{\text{root}}, pk, \phi, sk, usk, cred$ such that it holds that $\text{CheckShow}(pp, pk_{\text{root}}, pk, \phi, sk, usk, cred) = 1$ we require

$$\begin{aligned} &\text{output}_{\mathcal{A}}[\mathcal{A} \leftrightarrow \text{ShowProve}(pp, pk_{\text{root}}, pk, \phi, sk, usk, cred)] \\ \approx\ &\text{output}_{\mathcal{A}}[\mathcal{A} \leftrightarrow \mathfrak{S}_{\text{ShowProve}}(td, pk_{\text{root}}, pk, \phi)] \end{aligned}$$

2. *"Anonymity when delegating a credential"*: For all $pk_{\text{root}}, \vec{A}^*, d^*, pk^*, usk, usk_{\text{alt}}, cred, cred_{\text{alt}}$ with $\text{CheckDeleg}(pp, pk_{\text{root}}, usk, cred, \vec{A}^*) = \text{CheckDeleg}(pp, pk_{\text{root}}, usk_{\text{alt}}, cred_{\text{alt}}, \vec{A}^*) = 1$, we require:

$$\begin{aligned} &\text{output}_{\mathcal{A}}[\mathcal{A} \leftrightarrow \text{DelegIssue}(pp, pk_{\text{root}}, usk, cred, \vec{A}^*, d^*, pk^*)] \\ \approx\ &\text{output}_{\mathcal{A}}[\mathcal{A} \leftrightarrow \mathfrak{S}_{\text{DelegIssue}}(td, pk_{\text{root}}, \vec{A}^*, d^*, pk^*, usk_{\text{alt}}, cred_{\text{alt}})] \end{aligned}$$

3. *"Anonymity when receiving a credential"*: For all $pk_{\text{root}}, \vec{A}^*, d^*, pk^*, sk^*, usk^*$ such that $\text{CheckPseud}(pp, pk^*, sk^*, usk^*) = 1$, we require:

$$\begin{aligned} &\text{output}_{\mathcal{A}}[\mathcal{A} \leftrightarrow \text{DelegRcv}(pp, pk_{\text{root}}, \vec{A}^*, d^*, pk^*, sk^*, usk^*)] \\ \approx\ &\text{output}_{\mathcal{A}}[\mathcal{A} \leftrightarrow \mathfrak{S}_{\text{DelegRcv}}(td, pk_{\text{root}}, \vec{A}^*, d^*, pk^*)] \end{aligned}$$

4. *"Pseudonyms are unlinkable"*: for all ppt $\mathcal{A}$ there is a negligible function $negl$ such that for all $\lambda \in \mathbb{N}$,

$$\begin{aligned} \Pr[&(pp, osk, td) \leftarrow \mathfrak{S}^{pp}(1^\lambda), \\ &(usk_0, id_0), (usk_1, id_1) \leftarrow \text{KeyGen}(pp), \\ &b \leftarrow \{0,1\}, \\ &b' \leftarrow \mathcal{A}^{\mathcal{O}_{\text{FormNym}}(\cdot), \mathcal{O}_{\text{Open}}(\cdot)}(1^\lambda, pp, td, usk_0, id_0, usk_1, id_1); b = b'] \leq 1/2 + negl(\lambda) \end{aligned}$$

where $\mathcal{O}_{\text{FormNym}}(1^n)$ returns pseudonyms of usk_b, i.e. it first runs $(pk, sk) \leftarrow \text{FormNym}(pp, usk_b, 1^n)$ and then returns pk. $\mathcal{O}_{\text{Open}}(pk)$ returns $\perp$ if pk was previously output by $\mathcal{O}_{\text{FormNym}}(\cdot)$. Otherwise it returns $\text{Open}(pp, osk, pk)$. ◇

Definition D.3 (Soundness). Consider the following experiment $\text{Exp}^{\text{soundness}}_{\Pi,\mathcal{A}}(\lambda)$ for a DAAC system Π, and an adversary $\mathcal{A}$.

- The experiment runs $(pp, osk) \leftarrow \text{Setup}(1^\lambda)$.
- The experiment maintains a list of user keys and identities $((usk_1, id_1), (usk_2, id_2), \dots)$, which is lazily generated, i.e. whenever an oracle wants to access an index i for the first time, the corresponding entry is created as $(usk_i, id_i) \leftarrow \text{KeyGen}(pp)$.
- For each user $i \in \mathbb{N}$, the experiment maintains an initially empty key pair list $\mathbf{pk}_i$ (for the user's pseudonyms) and an initially empty credential list $\mathbf{cred}_i$ (for the user's credentials).
- $\mathcal{A}$ gets pp, osk and may adaptively query the following oracles for any $u, u^* \in \mathbb{N}$ and any parameters of his choice:

$\text{FormNym}^u(1^n)$ *makes user u create a new pseudonym*: It runs $(pk, sk) \leftarrow \text{FormNym}(pp, usk_u, 1^n)$, records the tuple on $\mathbf{pk}_u$ and returns pk.

CreateCred$^u(i)$ *makes user u create a root credential for his ith pseudonym*: It finds the ith entry (pk, sk) on $\mathbf{pk}_u$, runs $cred \leftarrow \text{CreateCred}(pp, pk, sk)$ and appends $(cred, pk_{\text{root}} = pk)$ to $\mathbf{cred}_u$.

DelegateCred$^u(j, \vec{A}^*, d^*, pk^*)$ *makes user u delegate a credential to $\mathcal{A}$*: It finds the jth entry $(cred, pk_{\text{root}})$ on $\mathbf{cred}_u$. It then runs the protocol $\text{DelegIssue}(pp, pk_{\text{root}}, usk_u, cred, \vec{A}^*, d^*, pk^*)$ interacting with $\mathcal{A}$.

ReceiveCred$^u(i, pk_{\text{root}}, \vec{A}^*, d^*)$ *makes user u receive a credential from $\mathcal{A}$*: It finds the ith entry (pk^*, sk^*) on $\mathbf{pk}_u$ and runs the protocol $cred^* \leftarrow \text{DelegRcv}(pp, pk_{\text{root}}, \vec{A}^*, d^*, pk^*, sk^*, usk_u)$ interacting with $\mathcal{A}$. If $cred^* \neq \perp$, it appends $(cred^*, pk_{\text{root}})$ to $\mathbf{cred}_u$ and returns 1. Otherwise it does not modify the list and returns 0.

DelegateHonest$^{u,u^*}(i^*, j, \vec{A}^*, d^*)$ *makes user u delegate a credential to user u^**: It looks up the i^*th entry (pk^*, sk^*) in $\mathbf{pk}_{u^*}$, and the jth entry $(cred, pk_{\text{root}})$ in $\mathbf{cred}_u$. It then runs

$$\begin{aligned}&\text{DelegIssue}(pp, pk_{\text{root}}, usk_u, cred, \vec{A}^*, d^*, pk^*)\\ &\leftrightarrow \text{DelegRcv}(pp, pk_{\text{root}}, \vec{A}^*, d^*, pk^*, sk^*, usk_{u^*}) \rightarrow cred^*\end{aligned}$$

If $cred^* \neq \perp$, it appends $(cred^*, pk_{\text{root}})$ to $\mathbf{cred}_{u^*}$ and returns 1. Otherwise it does not modify the list and returns 0.

Show$^u(i, j, \phi)$ *makes user u show a credential to $\mathcal{A}$*: It looks up the ith entry (pk, sk) from $\mathbf{pk}_u$ and the jth entry $(cred, pk_{\text{root}})$ from $\mathbf{cred}_u$. It runs $\text{ShowProve}(pp, pk_{\text{root}}, pk, \phi, sk, usk_u, cred)$ interacting with $\mathcal{A}$.

- Eventually, $\mathcal{A}$ outputs challenge values $(pk_{\text{root}}, pk, \phi)$ and the experiment runs the protocol $\text{ShowVrfy}(pp, pk_{\text{root}}, pk, \phi)$ interacting with $\mathcal{A}$.
- If ShowVrfy does not output 1, the experiment outputs 0 and aborts. Otherwise, the experiment checks the following conditions. If any of them hold, the experiment outputs 1. Otherwise, it outputs 0.
  - *The user or root issuer's identity cannot be traced*: $\text{Open}(pp, osk, pk) = \perp$ or $\text{Open}(pp, osk, pk_{\text{root}}) = \perp$.
  - *$\mathcal{A}$ was able to impersonate some honest user*: $\text{Open}(pp, osk, pk) = id_i$ for some honest user's identity id_i.
  - *$\mathcal{A}$ was able to show a credential he did not receive*: $\text{Open}(pp, osk, pk_{\text{root}}) = id_i$ for some honest user's id_i and $\mathcal{A}$ never queried DelegateCred$^u(j, \vec{A}^*, d^*, pk^*)$ for any $u \in \mathbb{N}$ where all of the following hold:
    * the jth entry on $\mathbf{cred}_u$ is $(\cdot, pk_{\text{root}})$ (i.e. the delegated credential has the same root as in the challenge)
    * $\vec{A}^* \succeq \phi$ (i.e. the delegated attributes cover the challenge predicate)
    * $\text{Open}(pp, osk, pk) = \text{Open}(pp, osk, pk^*)$ or $d^* = 1$ (i.e. the identity behind receiver and challenge pseudonym is the same or the received credential is marked delegatable)

Π is *sound* if for all ppt adversaries $\mathcal{A}$ there exists a negligible function *negl* with

$$\Pr[\text{Exp}^{\text{soundness}}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq negl(\lambda)$$

for all $\lambda \in \mathbb{N}$. $\diamond$

In this experiment, the adversary wins if (1) in the show protocol he is able to use a pseudonym that is traced to an honest user's identity, or (2) he can show a credential that he never received and that is rooted at a pseudonym pk_{root} that is traced to an honest user. In particular, (2) covers the case that pk_{root} *is* one of the honest user's pseudonyms (cf. correctness of Open). Note that in the experiment, we supply the opener's secret osk to $\mathcal{A}$, making the security guarantees apply to the opener in the system, too.

We note that for the sake of simplicity, our soundness definition does not necessarily guarantee security in contexts where protocols may be executed concurrently (standard techniques can be applied to gain security in such contexts).

E Security proofs for the delegatable attribute-based anonymous credential construction

We split the proof of Theorem 6.1 into three lemmas: Correctness, anonymity, and soundness.

Lemma E.1 (Correctness). *If $\mathcal{E}$, $\mathcal{S}$ and its protocol for deriving a signature on a committed value are correct, then Construction 6.1 is correct (Definition D.1).*

The proof is straight-forward.

Lemma E.2 (Anonymity). *If $\mathcal{S}$ is perfectly derivation private (Definition 4.3), its protocol for deriving a signature on a committed value is secure for the receiver (Definition A.1), and $\mathcal{E}$ is CCA-secure, then Construction 6.1 has anonymity (Definition D.2).*

Proof. We define $\mathfrak{S}^{pp}(1^\lambda)$ to run the credential system's $pp \leftarrow \text{Setup}(1^\lambda)$ honestly except that it embeds public parameters for the zero-knowledge argument of knowledge into pp such that it knows the simulation trapdoor $td_{\mathcal{ZK}}$ (e.g., [Dam00]). It outputs pp from Setup and the simulation trapdoor $td = (pp, td_{\mathcal{ZK}})$.
"Anonymity when showing a credential": The simulator $\mathfrak{S}_{\text{ShowProve}}(td, pk_{\text{root}}, pk, \phi)$ runs the zero-knowledge simulator for the *ZKAK* listed in the construction. Since the only messages sent by an honest prover are the ones from the zero-knowledge argument, $\mathfrak{S}_{\text{ShowProve}}$ perfectly simulates the view of any adversary $\mathcal{A}$.

"Anonymity when delegating a credential": The simulator $\mathfrak{S}_{\text{DelegIssue}}(td, pk_{\text{root}}, \vec{A}^*, d^*, pk^*, usk_{\text{alt}}, cred_{\text{alt}})$ first parses $cred_{\text{alt}}$ as $(\sigma_{\text{alt}}, mk_{\text{alt}}, \vec{A}_{\text{alt}}, d_{\text{alt}})$ and pk_{root} as $(pk_{\mathcal{S},\text{root}}, c_{\text{root}}, \gamma_{\text{root}})$. Then the simulator computes the anonymized credential as $(\sigma', mk') \leftarrow \text{Transform}(pp, \vec{m}^{(\vec{A}_{\text{alt}}, usk_{\text{alt}}, pk_{\text{root}})}, \vec{m}^{(\vec{A}^*, 0, pk_{\text{root}})}, \sigma, mk, I^{(1,\vec{A}^*)})$. Because of perfect derivation privacy, (σ', mk') is distributed exactly as in honest runs of the protocol. The simulator proceeds with (σ', mk') exactly as described in DelegIssue starting with Step 4.

"Anonymity when receiving a credential": The simulator $\mathfrak{S}_{\text{DelegRcv}}(td, pk_{\text{root}}, \vec{A}^*, d^*, pk^*)$ works as follows: For $d^* = 1$, the simulator just sets $sk^* = \perp$, $usk^* = 0$ and runs $cred^* \leftarrow \text{DelegRcv}(pp, pk_{\text{root}}, \vec{A}^*, d^*, pk^*, 0, 0)$ interacting with $\mathcal{A}$. It then locally outputs $cred^*$.[3] In this case, no messages are sent to $\mathcal{A}$, hence trivially the simulator behaves indistinguishably from the real protocol.

For $d^* = 0$, the simulator $\mathfrak{S}_{\text{DelegRcv}}$ receives K from $\mathcal{A}$ and computes $C = \text{Commit}_{\mathcal{S}}(K, 0, r)$

[3]The local output of the simulator is not relevant for anonymity, but will be useful for the soundness proof later.

for some random r (i.e. it implicitly sets $usk^* = 0$). It then sends C to $\mathcal{A}$ and uses the zero-knowledge simulator to simulate the proof protocol Step 5b.

Finally, $\mathfrak{S}_{\text{DelegRcv}}$ runs $(\sigma^*, mk^*) \leftarrow \text{BlindRcv}_{\mathcal{S}}(0, n+1, I^{(d^*, \vec{A}^*)}, K, C, r))$. Because BlindRcv has security for the receiver (Definition A.1), this produces the exact same view as in honest executions of DelegRcv. Overall, the view of $\mathcal{A}$ is the same as with the real protocol.

The simulator checks $\text{VrfyMk}_{\mathcal{S}}(pk_{\mathcal{S},\text{root}}, \vec{m}^{(\vec{A}^*, usk^*, pk_{\text{root}})}, \sigma^*, mk^*, I^{(d^*, \vec{A}^*)}) \overset{!}{=} 1$ as in Step 6 but with $usk^* = 0$ and, if it succeeds, locally outputs $cred^* = (\sigma^*, mk^*, \vec{A}^*, d^*)$.

"Pseudonyms are unlinkable": Note that the output of $\mathcal{O}_{\text{FormNym}}(\cdot)$ consists of a signature verification key $pk_{\mathcal{S}}$, which is independent of usk_b, id_b, a ciphertext $c \leftarrow \text{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(usk_b))$, and a signature of knowledge γ, which can be simulated independently of b. The $\mathcal{O}_{\text{Open}}$ oracle corresponds directly to a decryption oracle for ciphertexts c that were not created by $\mathcal{O}_{\text{FormNym}}$. Hence by CCA-security of $\mathcal{E}$ and a standard hybrid argument, one can easily see that no ppt algorithm has non-negligible advantage distinguishing whether the $\mathcal{O}_{\text{FormNym}}$ oracle encrypts $f(usk_0)$ or $f(usk_1)$. □

Lemma E.3 (Soundness)**.** *If Construction 6.1 has anonymity (Definition 3.2), $\mathcal{S}$ is unforgeable (Definition 4.4), perfect derivation private (Definition 4.3), and its protocol for deriving a signature on a committed value is secure for the issuer and secure for the receiver (Definitions A.2 and A.1), $\mathcal{OW}$ is a secure one-way function, $\mathcal{H}$ is a collision-resistant hash function, and $\mathcal{E}$ is correct (i.e. encryption is perfectly binding), then Construction 6.1 is* sound *(Definition D.3).*

Let $(pk_{\text{root}}, pk, \phi)$ be the challenge that $\mathcal{A}$ chooses. We write $pk_{\text{root}} = (pk_{\mathcal{S},\text{root}}, c_{\text{root}}, \gamma_{\text{root}})$. There are two types of ppt adversaries $\mathcal{A}$: Type 1 adversaries choose their challenge in one of the following ways:

- $\text{Open}(pp, osk, pk) = \perp$ or $\text{Open}(pp, osk, pk_{\text{root}}) = \perp$
- $\text{Open}(pp, osk, pk) = id_i$ for some honest user's identity id_i
- $\text{Open}(pp, osk, pk_{\text{root}}) = id_i$ for some honest user's identity id_i and $pk_{\mathcal{S},\text{root}}$ is not part of any pseudonym created through the FormNym oracle.

Type 2 adversaries choose their challenge such that $pk_{\mathcal{S},\text{root}}$ is part of some pseudonym created by the FormNym oracle, and $\mathcal{A}$ never queried the oracle $\text{DelegateCred}^u(j, \vec{A}^*, d^*, pk^*)$ where the credential to be delegated is also rooted at pk_{root}, $\vec{A}^* \succeq \phi$, and $\text{Open}(pp, osk, pk^*) = \text{Open}(pp, osk, pk) \vee d^* = 1$.

Any adversary with non-negligible advantage against the soundness experiment gives rise to a Type 1 or a Type 2 adversary with non-negligible advantage against the soundness experiment. We will show that Type 1 adversaries invert the one-way function and Type 2 adversaries forge signatures.

Type 1 adversaries. Let $\mathcal{A}$ be a Type 1 adversary and let q be an upper bound for the number of honest users $\mathcal{A}$ creates in the soundness experiment. Without loss of generality, we assume that $\mathcal{A}$ only queries user indices $1, \ldots, q$. We construct an adversary $\mathcal{B}$ against the one-way function.

$\mathcal{B}$ gets public parameters $pp_{\mathcal{OW}}$, a description of $f : M_{\mathcal{S}} \to M_{\mathcal{E}}$, and some $y \in M_{\mathcal{E}}$ as input. It completes the setup by running $(pk_{\mathcal{E}}, sk_{\mathcal{E}}) \leftarrow \text{KeyGen}_{\mathcal{E}}(pp_{\mathcal{OW}})$. It hands $pp = (pp_{\mathcal{OW}}, pk_{\mathcal{E}}, f)$ and the opening key $osk = sk_{\mathcal{E}}$ to $\mathcal{A}$.

$\mathcal{B}$ chooses a random user $u' \leftarrow \{1, \ldots, q\}$ and implicitly sets $id_{u'} = y$ and $usk_{u'} := usk := f^{-1}(y)$. In the following, all credentials for user u' will be stored with respect to the user secret 0, while all pseudonyms will use usk.

$\mathcal{B}$ can answer all of $\mathcal{A}$'s oracle queries honestly except for those that involve user u'. $\mathcal{B}$ answers $\mathrm{FormNym}^{u'}$ queries by encrypting y and simulating the signature of knowledge γ. For $\mathrm{CreateCred}^{u'}$, it simply uses 0 as a user secret in the signature. For $\mathrm{Show}^{u'}$, $\mathcal{B}$ aborts if the specified credential's attributes do not fulfill the given formula ϕ. Otherwise, it runs $\mathfrak{S}_{\mathrm{ShowProve}}(td, pk_{\mathrm{root}}, pk, \phi)$ with $\mathcal{A}$. For $\mathrm{DelegateCred}^{u'}$, $\mathcal{B}$ possesses the relevant credential cred with embedded user secret 0. $\mathcal{B}$ sets $\mathit{cred}_{\mathrm{alt}} = \mathit{cred}$ and $\mathit{usk}_{\mathrm{alt}} = 0$. It runs $\mathfrak{S}_{\mathrm{DelegIssue}}(td, pk_{\mathrm{root}}, \vec{A}^*, d^*, pk^*, \mathit{usk}_{\mathrm{alt}}, \mathit{cred}_{\mathrm{alt}})$ with $\mathcal{A}$, which produces the expected view.

For $\mathrm{ReceiveCred}^{u'}(i, pk_{\mathrm{root}}, \vec{A}^*, d^*)$, $\mathcal{B}$ runs $\mathit{cred}^* \leftarrow \mathfrak{S}_{\mathrm{DelegRcv}}(td, pk_{\mathrm{root}}, \vec{A}^*, d^*, pk^*)$ (from the proof of Lemma E.2) interacting with $\mathcal{A}$. If $\mathit{cred}^* \neq \perp$, it adds cred^* (which, by construction of $\mathfrak{S}_{\mathrm{DelegRcv}}$ above, is a credential for $\mathit{usk}^* = 0$) to u''s list of credentials.

Overall, the view of $\mathcal{A}$ is exactly as in the actual experiment. Eventually, $\mathcal{A}$ outputs $(pk_{\mathrm{root}}, pk, \phi)$ and runs the show protocol with $\mathcal{B}$. If $\mathcal{A}$ does not succeed in it, $\mathcal{B}$ aborts. We distinguish three cases (at least one of which must happen in order for $\mathcal{A}$ to be a successful Type 1 adversary):

Case 1: $\mathrm{Open}(pp, osk, pk) = \perp$ or $\mathrm{Open}(pp, osk, pk_{\mathrm{root}}) = \perp$. This happens only with negligible probability as the argument of knowledge in the show protocol and the signature of knowledge in pk_{root} guarantee that pk and pk_{root} contain valid ciphertexts, respectively.

Case 2: $\mathrm{Open}(pp, osk, pk) = id_i$ for some honest user's identity id_i. With probability at least $1/q$, $id_i = y$ and so pk contains an encryption c of y. We use the extractor for the argument of knowledge protocol inside the show protocol to extract (among others) usk with $f(\mathit{usk}) = y$. $\mathcal{B}$ then outputs usk, inverting the one-way function.

Case 3: $\mathrm{Open}(pp, osk, pk_{\mathrm{root}}) = id_i$ for some honest user's id_i but $pk_{\mathrm{root}} = (pk_{\mathcal{S},\mathrm{root}}, c_{\mathrm{root}}, \gamma_{\mathrm{root}})$ such that $pk_{\mathcal{S},\mathrm{root}}$ is not part of any honest user's pseudonym. In this case, with probability at least $1/q$, $id_i = y$ and so pk_{root} contains an encryption c of y and a valid signature of knowledge γ (as guaranteed by the CheckShow check within the ShowVrfy protocol). Note that $\mathcal{B}$ never had to simulate a signature of knowledge on $(pk_{\mathcal{S},\mathrm{root}}, c)$. Consequently from the signature of knowledge forgery, we can extract (among others) usk with $f(\mathit{usk}) = y$. $\mathcal{B}$ then outputs usk, inverting the one-way function. $\square$

We now turn to Type 2 adversaries.

Type 2 adversaries. Let $\mathcal{A}$ be a Type 2 adversary and let q be an upper bound for the number of queries to the FormNym oracle. We construct $\mathcal{B}$ against the DMS scheme's unforgeability.

$\mathcal{B}$ receives $pp_{\mathcal{S}}, pk_{\mathcal{S}}$ from the unforgeability experiment. It completes the setup by running $(pk_{\mathcal{E}}, sk_{\mathcal{E}}) \leftarrow \mathrm{KeyGen}_{\mathcal{E}}(pp_{\mathcal{S}})$, $f \leftarrow \mathrm{GenFnct}_{\mathcal{OW}}(pp_{\mathcal{S}})$. It hands $pp = (pp_{\mathcal{S}}, pk_{\mathcal{E}}, f)$ and the opening key $osk = sk_{\mathcal{E}}$ to $\mathcal{A}$.

$\mathcal{B}$ then chooses $k \leftarrow \{1, \ldots, q\}$. It sets $pk^*_{\mathrm{root}} := (pk_{\mathcal{S}}, c, \gamma)$ for $\mathrm{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(\mathit{usk}_u); r') = c$ with random r' and $\gamma = \mathit{NIZK}[(\mathit{usk}_u, r'); c = \mathrm{Encrypt}_{\mathcal{E}}(pk_{\mathcal{E}}, f(\mathit{usk}_u); r')](pk_{\mathcal{S}}, c)$. When $\mathcal{A}$ calls $\mathrm{FormNym}^u(1^n)$ for the kth time, $\mathcal{B}$ returns pk^*_{root}.

For later, $\mathcal{B}$ chooses a random $r \leftarrow M_{\mathcal{S}}$ and queries its signature oracle for $(0, \ldots, 0, r) \in \mathbb{Z}_p^{n+2}$ with index set $I = \{n+1\}$ (i.e. the last 0, whose position corresponds to the user secret in credential signatures, can be changed and r cannot). We call the resulting signature σ_{dummy} and the malleability key $\mathit{mk}_{\mathrm{dummy}}$.

$\mathcal{B}$ can answer $\mathcal{A}$'s oracle queries honestly, except for the ones involving pk^*_{root}. For all credentials rooted at pk^*_{root}, $\mathcal{B}$ does *not* run the $\mathcal{S}$ algorithms and simply stores them as $(\sigma = \epsilon, \mathit{mk} = \epsilon, \vec{A}, d)$ (i.e. without signatures or malleability keys) on honest users' lists $\mathbf{cred}_u$. When $\mathcal{A}$ queries the Show oracle, $\mathcal{B}$ uses the simulator $\mathfrak{S}_{\mathrm{ShowProve}}$.

When $\mathcal{A}$ queries $\mathrm{DelegateCred}^u(j, \vec{A}^*, d^*, pk^*)$ where j refers to a credential rooted at pk^*_{root}, then there are two cases:

If $d^* = 1$, then $\mathcal{B}$ simply queries its signature oracle for $\vec{m}^{(\vec{A}^*,0,pk^*_{\text{root}})}, I^{(d^*,\vec{A}^*)}$ to receive σ', mk', and sends those to $\mathcal{A}$ (because of perfect derivation privacy, σ', mk' will have exactly the distribution that $\mathcal{A}$ expects).
If $d^* = 0$, then $\mathcal{B}$ runs $(K, k) \leftarrow \text{BlindInit}_{\mathcal{S}}(\sigma_{\text{dummy}}, mk_{\text{dummy}}, (0, \ldots, 0, r), n + 1)$ and sends K to $\mathcal{A}$ (note that the protocol's security for the issuer guarantees that K has the expected distribution). $\mathcal{A}$ responds with a commitment C, then engages in the argument of knowledge protocol in Step 5b proving he can open the commitment. Using the extractor for the protocol, $\mathcal{B}$ extracts usk^* and r such that $C = \text{Commit}_{\mathcal{S}}(K, usk^*, r)$ and pk^* contains an encryption of $f(usk^*)$. $\mathcal{B}$ then queries its signature oracle for $\vec{m}^{(\vec{A}^*,usk^*,pk^*_{\text{root}})}$ and index set $I^{(d^*,\vec{A}^*)}$, receiving (σ^*, mk^*). It runs the simulator $\mathfrak{S}_{\text{BlindIssue}}(usk^*, n+1, I^{(d^*,\vec{A}^*)}, K, C, r, \vec{m}^{(\vec{A}^* usk^*,pk^*_{\text{root}})}, \sigma^*, mk^*)$ (Definition A.2) interacting with $\mathcal{A}$.

Overall, the view of $\mathcal{A}$ is consistent with what it would see if $\mathcal{B}$ had implemented the oracles correctly. Eventually, $\mathcal{A}$ outputs the challenge $(pk_{\text{root}}, pk, \phi)$ such that $pk_{\text{root}} = (pk_{\mathcal{S},\text{root}}, c_{\text{root}}, \gamma_{\text{root}})$ where $pk_{\mathcal{S},\text{root}}$ is part of some pseudonym created by the FormNym oracle, and $\mathcal{A}$ never queried $\text{DelegateCred}^u(j, \vec{A}^*, d^*, pk^*)$ where the credential to be delegated is also rooted at pk_{root}, $\vec{A}^* \succeq \phi$, and $\text{Open}(pp, osk, pk^*) = \text{Open}(pp, osk, pk) \vee d^* = 1$.

With probability at least $1/q$, $pk_{\mathcal{S},\text{root}}$ is part of pk^*_{root}. Otherwise, $\mathcal{B}$ aborts. If $\mathcal{B}$ does not abort, $\mathcal{A}$ engages $\mathcal{B}$ in the argument of knowledge inside the ShowVrfy protocol. We use the extractor from the argument of knowledge to extract a witness $(usk, r', \sigma', \vec{A})$ where σ' is a signature on $\vec{m}^{(\vec{A},usk,pk_{\text{root}})}$ under $pk_{\mathcal{S}}$, and $\phi(\vec{A}) = 1$. $\mathcal{B}$ outputs $\vec{m}^{(\vec{A},usk,pk_{\text{root}})}, \sigma'$ as a forgery.

It remains to show that $\mathcal{B}$ never made a signature query for $\vec{m}', I'$ where $\vec{m}' \equiv_{I'} \vec{m}^{(\vec{A},usk,pk_{\text{root}})}$. For contradiction, assume that there is some $(\vec{m}', I')$ which $\mathcal{B}$ queried at some point. We go through the three cases where this could have happened.

(1) Assume that $(\vec{m}', I')$ was queried by $\mathcal{B}$ in the beginning of the experiment to receive σ_{dummy}. That means that $\vec{m}' = (0, \ldots, 0, r)$ and $I' = \{n+1\}$. Since $\mathcal{A}$'s view is independent of r (cf. Definition A.2), with overwhelming probability $r \neq \mathcal{H}(pk_{\text{root}})$. Hence $\vec{m}' \not\equiv_{I'} \vec{m}^{(\vec{A},usk,pk_{\text{root}})}$, contradicting the assumption.

(2) Assume that $(\vec{m}', I')$ was queried when $\mathcal{A}$ requested $\text{DelegateCred}^u(j, \vec{A}^*, d^*, pk^*)$ with $d^* = 1$. That means that $\vec{m}' = \vec{m}^{(\vec{A}^*,0,pk^*_{\text{root}})}$, $I' = I^{(d^*,\vec{A}^*)}$. Then because $\vec{m}' \equiv_{I'} \vec{m}^{(\vec{A},usk,pk_{\text{root}})}$, it must follow that $\vec{A}^* \succeq \vec{A} \succeq \phi$ and $\mathcal{H}(pk_{\text{root}}) = \mathcal{H}(pk^*_{\text{root}})$. Collision resistance of $\mathcal{H}$ implies that with overwhelming probability, $pk_{\text{root}} = pk^*_{\text{root}}$. Hence $\mathcal{A}$ queried DelegateCred for a credential rooted at pk_{root} with $\vec{A}^* \succeq \phi$ and $d^* = 1$, meaning $\mathcal{A}$ would have lost the experiment.

(3) Assume that $(\vec{m}', I')$ was queried when $\mathcal{A}$ requested $\text{DelegateCred}^u(j, \vec{A}^*, d^*, pk^*)$ with $d^* = 0$. That means that $\vec{m}' = \vec{m}^{(\vec{A}^*,usk^*,pk^*_{\text{root}})}$, $I' = I^{(d^*,\vec{A}^*)}$. As in (2), it follows that with overwhelming probability, $\vec{A}^* \succeq \vec{A} \succeq \phi$ and $pk_{\text{root}} = pk^*_{\text{root}}$. Additionally, $usk = usk^*$ because $\vec{m}' \equiv_{I'} \vec{m}^{(\vec{A},usk,pk_{\text{root}})}$ and $n + 1 \notin I'$. It follows that $\text{Open}(pp, osk, pk^*) = f(usk^*) = f(usk) = \text{Open}(pp, osk, pk)$ because $\mathcal{B}$ extracted usk and usk^* from arguments of knowledge guaranteeing that the encryptions in pk and pk^* are of $f(usk)$ and $f(usk^*)$, respectively. Hence $\mathcal{A}$ queried DelegateCred for a credential rooted at pk_{root} with $\vec{A}^* \succeq \phi$ and $\text{Open}(pp, osk, pk^*) = \text{Open}(pp, osk, pk)$, meaning $\mathcal{A}$ would have lost the experiment.

Overall, if $\mathcal{A}$ does not lose the experiment, then with overwhelming probability $\mathcal{B}$ outputs a valid forgery for $\mathcal{S}$. $\square$