Practical Delegatable Anonymous Credentials From Equivalence Class Signatures

Omid Mir[1], Daniel Slamanig[2], Balthazar Bauer[3], and René Mayrhofer[1]

[1] LIT Secure and Correct Systems Lab, Institute of Networks and Security, Johannes Kepler University Linz, Austria
mir@ins.jku.at, rm@ins.jku.at
[2] AIT Austrian Institute of Technology, Vienna, Austria
daniel.slamanig@ait.ac.at
[3] IRIF, Université de Paris Cité, France
Balthazar.Bauer@ens.fr

Abstract. Anonymous credentials systems (ACs) are a powerful cryptographic tool for privacy-preserving applications and provide strong user privacy guarantees for authentication and access control. ACs allow users to prove possession of attributes encoded in a credential without revealing any information beyond them. A delegatable AC (DAC) system is an enhanced AC system that allows the owners of credentials to delegate the obtained credential to other users. This allows to model hierarchies as usually encountered within public-key infrastructures (PKIs). DACs also provide stronger privacy guarantees than traditional AC systems since the identities of issuers and delegators are also hidden. A credential issuer's identity may convey information about a user's identity even when all other information about the user is protected.

We present a novel delegatable anonymous credential scheme that supports attributes, provides anonymity for delegations, allows the delegators to restrict further delegations, and also comes with an efficient construction. In particular, our DAC credentials do not grow with delegations, i.e., are of constant size. Our approach builds on a new primitive that we call structure-preserving signatures on equivalence classes on updatable commitments (SPSEQ-UC). The high-level idea is to use a special signature scheme that can sign vectors of set commitments which can be extended by additional set commitments. Signatures additionally include a user's public key, which can be switched. This allows us to efficiently realize delegation in the DAC. Similar to conventional SPSEQ signatures, the signatures and messages can be publicly randomized and thus allow unlinkable showings in the DAC system. We present further optimizations such as cross-set commitment aggregation that, in combination, enable selective, efficient showings in the DAC without using costly zero-knowledge proofs. We present an efficient instantiation that is proven to be secure in the generic group model and finally demonstrate the practical efficiency of our DAC by presenting performance benchmarks based on an implementation.

Keywords: Equivalence-class signatures, set commitments, delegatable anonymous credentials

1 Introduction

Anonymous credentials (ACs) [CL03, CL04, PS16, ILV11, San20, FHS19, HS21, CLP22] provide a means to strong authentication with built-in authorization (access control) and play an essential role in privacy-preserving applications. In such a system credentials encode a set of attributes and are issued by some trusted issuer(s). A credential holder can then show a credential to a verifier by

selectively revealing attributes (or proving more general relations about hidden attributes) in a way that multiple showings of the same credential cannot be linked. This should hold even if issuers and verifiers collude and ideally even when the issuer is allowed to generate its key material maliciously. Technologies related to ACs found numerous applications such as PrivacyPass [DGS+18] by Cloudflare (available as extensions for Chrome and Firefox), an enhanced variant by Google [KLOR20] being integrated into the Trust Tokens API[4] or the PrivateStats proposal by Facebook.[5] A recent large scale real-world application of ACs is the realization of private groups within the popular Signal messenger [CPZ20].

When using credentials in our daily lives, we frequently run into a difficult problem: *delegation*. We often need to delegate our tasks, responsibilities, and permissions to someone else, or we simply want to share our access to resources and services with another person or among our different electronic devices. Indeed, in practice, credentials are usually issued in a hierarchical manner, e.g., in a public key infrastructure (PKI) there is a chain of certificates between the user certificate and a trusted root authority. An easy way to achieve delegation is credential sharing. However, giving away the full and unlimited power of a credential is often not what we want. Consider the example of a manager of a company who wants to delegate a task that requires filling and signing of documents to their secretary. Providing the secretary with the signing keys gives away the power to sign arbitrary documents in the name of the manager – which is obviously not what the manager intended. Even worse, since the actual signing key was shared, the only way to revoke these powers from the secretary is to revoke and invalidate the whole credential. This problem clearly carries over to anonymous credentials.

As also highlighted in [BCC+09, CDD17, CL19], traditional anonymous credentials in which we assume that the verifying party (or a service provider) knows the public key of the credential issuers do not provide strong privacy when used in a hierarchical structure.[6] That is, this chain of issuers may reveal sensitive information about the issuer's organizational structure or the credential holder. Consequently, in this setting it is desirable to provide anonymity guarantees even if there is a certain type of collusion of issuers and/or delegators.

Delegatable anonymous credentials. Chase and Lysyanskaya in [CL06] introduced the notion of delegatable anonymous credentials (DACs). DAC schemes, later improved in [BCC+09], are particularly interesting for applications such as (physical) access control [MSM+18], root of trust [CL19], or authorizing transactions in permissioned blockchains [CDD17, BCET21]. Following the notion of levels by Belenkiy et al. [BCC+09], an initial credential is issued for a level $L = 1$. Any level L credential can then be used to delegate a level $L+1$ credential to another user. As with traditional anonymous credentials, users can then show a credential (i.e., prove possession of their credential) without disclosing their identity and without revealing anything about non-disclosed attributes to a verifying party. Thereby, only the identity (i.e., public key) of the root issuer but none of the intermediate levels is revealed during the verification. The construction of [BCC+09] is based on a commitment scheme and a signature scheme with randomizable non-interactive zero-knowledge (NIZK) proofs. This approach can be instantiated from Groth-Sahai (GS) commitments and GS NIZK proofs [GS08] resulting in credentials and showings of size linear in the chain length L. Unfortunately, using tools like GS proofs make this scheme inefficient for practical use as the quite expensive statements result in poor computational performance and large credential size. Several other DAC constructions have been proposed afterwards, e.g. [CKLM13, Fuc11, CKLM14],

[4] https://web.dev/trust-tokens/

[5] https://research.fb.com/privatestats

[6] Concealing the credential issuer has recently also been shown to have value in a setting of anonymous credentials without credential delegation [BEK+21, CLP22].

which follow roughly the same techniques as [BCC+09], i.e., using malleable proof systems (based on GS) as the main building block, and thus have similar performance characteristics.

Camenisch et al. [CDD17] propose a much more practical approach towards DAC in which one can prove possession of a credential chain in a privacy-preserving manner, but one cannot obtain credentials anonymously. Indeed, credential holders can see all attributes and public keys on all levels in plain, i.e., not offering an anonymous delegation phase. They present an efficient instantiation of their DAC scheme based on the structure-preserving signature (SPS) by Groth [Gro15] (we provide a comparison with [CDD17] in Appendix B). This approach has recently been integrated into Hyperledger Fabric [BCET21]. Later Blömer and Bobolz (BB) [BB18] proposed another practical DAC approach using dynamically malleable signatures (DMS) and NIZK proofs, which conceptually is similar to the approach in [CKLM14]. A DMS can sign messages and "placeholders", where during delegation "placeholders" can be replaced by concrete messages and thus enabling to add attributes during delegation in the DAC. Unfortunately, [BB18] only details the underlying signature scheme, which is based on Pointcheval-Sanders signatures [PS16], but the remaining parts of their generic constructions are not detailed, making concrete performance estimates hard. But since their showing must conceal the DMS signature and prove the verification relation of the DMS (resulting in a size linear in the number of undisclosed attributes), this imposes a rather complex NIZK statement. Finally, BB requires a trusted setup (and more particular a trapdoor in their parameters), something that is important to avoid in privacy-preserving primitives.

DAC using equivalence-class signatures. Crites and Lysyanskaya [CL19] provide probably the most efficient and conceptually most simple construction of DACs. Their construction does not rely on any computationally complex systems such as NIZK proofs. The main building block of their construction is a new type of signature scheme, called a *mercurial signature*. It extends structure-preserving signatures on equivalence classes (SPSEQ) [HS14, FHS19] to equivalence classes on the key space. SPSEQ in addition to randomizing signatures also provides randomization of signed messages (modeled as equivalence classes). Thus, SPSEQ allow similar applications as SPSs, but unlike the latter they do not need NIZK proofs on top, thereby yielding more efficient schemes. Mercurial signatures extend SPSEQ in the sense that it adds the property of transforming public keys into an equivalent one, i.e., additionally supports randomization of public keys.

Unfortunately, existing DAC schemes based on mercurial signatures [CL19, CL21] have some shortcomings: 1) They do not support attributes in a meaningful way. [CL19] does not support attributes at all and while Crites and Lysyanskaya in [CL21] show how to support attributes, it is only possible in a way where *all* attributes are *always* revealed during showings and thus no selective disclosure is supported. This feature is not very attractive for privacy-preserving applications. 2) Bauer and Fuchsbauer [BF20] point out a significant drawback of the weak form of anonymity provided by their DAC construction. That is, if Alice delegates a credential to Bob, she can identify Bob whenever he shows the credential, which indicates a severe violation of Bob's privacy. This is because, when Alice delegates a credential to Bob, she uses her secret key to sign Bob's pseudonym under her pseudonym (the randomized public key), which becomes part of Bob's credential (see [BF20] for more details). Moreover, 3) similar to [CDD17], the credential size depends on the delegation chain length L, and consequently, the size of the credential grows linearly with L.

Summarizing, the state-of-the-art in existing DAC schemes is that the ones that are conceptually simple and practically efficient do not provide all the desirable properties of supporting selective showing of attributes, being compact, and providing sufficiently strong anonymity guarantees at the same time.

1.1 Our Contribution

Our contribution is to formalize and present a construction and implementation of DAC that mitigate the aforementioned problems:

Delegatable Anonymous Credentials. We propose a novel delegatable anonymous credentials scheme (DAC). Our scheme provides the following key characteristics: *i*) It represents a *simple, and practical* construction without requiring zero-knowledge proofs (for complex statements), which makes it well-suited for real-world applications. *ii*) It is *constant-size* in two aspects. First, the bandwidth required for the credential showing protocol is independent of the number of the attributes, but only depends on the delegation depth. Second, unlike the schemes in [CL19, CDD17] and similar to [BB18] the credential size is independent of the length of the credential (delegation) chain. *iii*) Credentials are *attribute-based* in a sense that every level in the delegation chain is associated with a set of attributes that are certified by the respective delegator. Credential holders can then decide for every level whether and which attributes should be selectively revealed during a showing of a credential. Moreover, every delegator can *restrict delegation* in how many further levels can be delegated and whether attributes associated to previous levels should be valid (showable) or invalidate them (making them unshowable). *iv*) It provides *strong anonymity* which means that it not only supports an anonymous showing phase but also provides an anonymous delegation phase under a reasonable corruption model and anonymity holds even for maliciously generated issuer keys. Finally, *v*) our DAC comes with an *implementation* and evaluation that demonstrates its practical efficiency.

Novel Building Block. Our DAC scheme is based on a novel cryptographic building block that we call structure-preserving signatures on equivalence classes on updatable commitments (SPSEQ-UC). This primitive draws inspiration from structure-preserving signatures on equivalence classes (SPSEQ) [HS14, FHS19] as well as the set commitment scheme in [FHS19]. Loosely speaking their idea is to use SPSEQ to sign a randomizable set commitment and showing a credential amounts to randomizing the message (set commitment), randomizing and adapting the signature to the new message and providing the signature, randomized commitment and an opening to the randomized commitment. While in SPSEQ the message space is simply group element vectors, in SPSEQ-UC the message space is viewed as a vector (of length at most ℓ as an upper bound for the message vector) of randomizable set commitments. This concept is somewhat similar to signatures on randomizable ciphertexts (SoRC) [BFPV11, BF20]. However, in contrast to SoRC which does not allow to reveal a subset of the encrypted message, here it is also possible to reveal only a subset of the committed values of each commitment in the vector while guaranteeing the privacy of the non-revealed ones. Thereby, SPSEQ-UC needs to be unlinkable, which means the same commitment-signature pair can be revealed multiple times without being linkable to each other. One key feature is that SPSEQ-UC allows to extend signed vectors by additional set commitments. More precisely, in SPSEQ-UC signing of a commitment vector of length k also produces an update key $\mathsf{uk}_{k'}$ corresponding to an integer k' with $k \leq k' \leq \ell$. Given the update key $\mathsf{uk}_{k'}$ one can update a commitment vector $\boldsymbol{C}$ to a vector $\boldsymbol{C}'$ (i.e., extending it). Another key feature is that in a SPSEQ-UC scheme the signing process is tied to a user public key. It allows a signer to bind a signature to a given user public key such that this signature can then be adapted into another valid signature for a new user public key by anyone knowing the corresponding old user secret key.

We provide a rigorous security model for SPSEQ-UC which carefully crafts privacy notions similar to SPSEQ [FHS19] in order to guarantee that adapted (i.e., re-randomized) signatures, signatures after extending commitment vectors as well as signatures after switching user public-keys are distributed identically to new signatures and thus are all unlinkable to fresh signatures. This is

important for our application in DAC and other potential applications in privacy-preserving protocols. Moreover, we provide a provably secure construction of an SPSEQ-UC based on the SPSEQ scheme in [FHS15], which is proven secure in the generic group model, and the set commitment scheme in [FHS19]. We chose this path as our main focus is on efficiency, but we consider constructing SPSEQ-UC from standard assumptions, i.e., relying on the recent approach in [CLP22](building upon and improving the SPSEQ in [KSD19]), as an interesting avenue for future work.[7] Another direction would also be to adopt the modified set commitment scheme in [CLP22], which in addition to selective disclosure also supports non-membership proofs for disjoint sets, which would directly increase the expressiveness of the DAC scheme.

1.2 High Level Idea of Our Approach

On a very high level, our approach to construct DAC takes inspiration from the anonymous credentials in [FHS19] as well as the approach based on DMS in [BB18] but in contrast to the latter and similar to [CL19, CL21] avoids the use of NIZK for complex statements.

The idea in our DAC, omitting some details for the sake of brevity, is that in a hierarchy of delegations the root authority issues a SPSEQ-UC signature on a vector of commitments of length two, where the first one is a dummy commitment that is there for technical reasons and not assigned any or simpy some fixed attributes. The second commitment in the vector carries the attributes for the first delegatee and the public key included in the signature is the one of the delegatee. The delegatee if allowed to perform further delegations is then given an update key for this signature. This update key allows to further extend the commitment vector and thus delegating a credential for the next level in the delegation hierarchy. Again the public key of the delegatee, now playing the role of the delegator, is switched to the one of the next delegatee. This process keeps on going until the end of the delegation chain is reached (if no further delegations are allowed, then no update key is provided). One issue that is worth mentioning is that every delegator can control how far delegations can go by further restricting the update key and a delegator can also restrict the possibility to show attributes from a certain level in the hierarchy (which corresponds to a commitment in the commitment vector) by not providing the opening of the commitment to the delegatee.

Now showing a credential simply amounts to adapting the signature to a re-randomized signature for a re-randomized commitment vector and providing subset openings of the respective commitments. As we show in Section 3.4 we can realize a cross-commitment aggregation technique to make the opening of multiple commitments compact. Due to the properties of SPSEQ-UC such showings are unlinkable. More so, the properties of SPSEQ-UC also allow to realize an anonymous delegation process so that delegations cannot be tracked and all credentials in a delegation are indistinguishable from fresh SPSEQ-UC signatures on vectors of identical length.

1.3 Comparison with Previous Work

In Table 1, we provide a comparison of our approach with other existing efficient DAC schemes in the literature [BB18, CDD17, CL19, CL21]. We compare our DAC with these schemes in terms of the following criteria: **Attr** which means whether credentials include attributes that can at least be selectively revealed. We use $\approx$ to indicate that [CL21] supports attributes, but as all attributes

[7] However, there is an inherent issue when relying on the constructions in [KSD19, CLP22]. To cope with a malicious issuer in the DAC, something that we consider, they require a trusted generation of a common reference string (CRS). Knowledge of the respective trapdoor could be exploited to break anonymity.

always need to be revealed it does not support selective disclosure. **Expr** which compares the expressiveness of the supported showing policies, where R stands for arbitrary computable relations over attributes and S denotes the selective disclosure of a subset of attributes. We note that by avoiding NIZK proofs for complex statements, it seems necessary to be restricted to selective disclosure (S), which is however sufficient for most practical applications. **Rest** means whether it is possible to apply a restriction on the delegator's power during the delegation. Here, our scheme allows such restrictions in which a (superior) delegator can decide i) how many additional levels of delegation can be made, ii) to make all attributes of selected levels "invalid" by not providing the opening of the respective commitments, and, iii) how many attributes in each level (commitment) can be delegated by setting (potentially removing) key components in $\mathsf{uk}_{k'}$. BB [BB18] provides a type of restriction on the attributes such that delegators can prevent changing some attributes during delegation. However, one still can use these attributes in showings of a credential. Also, BB does not have a delegation-level concept and thus one can not control and restrict the delegation-level number (power). (**SAnon**) refers to strong anonymity guarantees meaning that no one can trace or learn information about the user's identity or anything beyond what they suppose to show during both the issuing/delegation and showing of credentials. Moreover, anonymity holds without relying on a trusted setup (and thus a potential trapdoor breaking anonymity) as well as the assumption of a malicious key generation by a (corrupted) root authority and user's key leaks [8]. Here ● means that the respective scheme satisfies all of these conditions, and ◐ means that it loses one or more of them.

Table 1. Comparison of practical DAC schemes (L: Delegation chain depth; n: Attributes; u: Undisclosed attributes).

Scheme	**Attr**	**Expr**	**Rest**	**SAnon**	**|Cred|**	**|Show|**
BB [BB18]	✓	S/R	$\approx$	◐†	$O(1)$	$O(u)$
CDD [CDD17]	✓	S/R	$\times$	◐$^{\clubsuit}$	$O(nL)$	$O(uL)$
CL [CL21]	$\approx$	$\times$	$\times$	◐*	$O(nL)$	$O(uL)$
Ours	✓	S	✓	●‡	$O(1)$	$O(L)$

† Requires a trusted setup and have a trapdoor associated to their parameters.
$^{\clubsuit}$ It does not support an anonymous delegation phase.
‡ We consider a malicious issuer key CA and all delegators keys can be exposed.
* It also allows an adversarial CA but no delegators's keys leaks.

With **|Cred|**, we denote the size of the credential. L indicates the length of the delegation chain. As it turns out, BB [BB18] (2 group elements) and our scheme (5 group elements) provide constant-size credentials. But as already mentioned, our scheme provides a simpler construction by *avoiding potentially costly linear-sized (in the number of attributes) zero-knowledge proofs*. With **|Show|**, we denote the size of the credential showing. Our showing is more efficient as it needs only a constant number of group elements (5 elements) and the commitment vector with the size of delegation L. BB needs to send the signature (2 elements) and the elements of proving knowledge

[8] We note that CL and our model require a credential cred_b of the anonymity challenge is on a delegation path from a (corrupted) root credential where all delegations have been performed honestly. However, we additionally allow the adversary to access the user corruption oracle in which we reveal the (delegators) user's secret keys to the adversary. CL cannot support this type of corruption as then the anonymity of their construction breaks down. This makes our model stronger than the one of CL.

of undisclosed attributes ($|\mathsf{ZKPoK}| = u$). Note that for practical use-cases, we assume $L < u$. For other schemes, this cost is much higher as their credentials grow linearly in the number of attributes and L.

2 Preliminaries and Notation

For a relation $\mathcal{R}$ let $[x]_{\mathcal{R}} = \{y|\mathcal{R}(x,y)\}$. If $\mathcal{R}$ is an equivalence relation, then $[x]_{\mathcal{R}}$ denotes the equivalence class of which x is a representative. We mention that a relation $\mathcal{R}$ is parameterized if it is well-defined as long as some other parameters are well-defined. Let $\mathbb{G}_1 = \langle P \rangle$, $\mathbb{G}_2 = \langle \hat{P} \rangle$, and $\mathbb{G}_T$ be groups of prime order p. A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a map, where it holds for all $(A, \hat{B}, a, b) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{Z}_p^2$ that $e(A^a, \hat{B}^b) = e(A, \hat{B})^{ab}$, and $e(P, \hat{P}) = g_T \neq 1_{\mathbb{G}_T}$, and e is efficiently computable. We will write $\mathbb{G}_i^* = \mathbb{G}_i \setminus \{1_{\mathbb{G}_i}\}$ and use $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P}) \leftarrow \mathsf{BGGen}(1^\lambda)$ to denote a bilinear group generator where p is a prime of bitlength λ. Given a finite set S, we denote by $x \leftarrow S$ the sampling of and an element uniformly at random in S. For an algorithm A, let $y \leftarrow \mathsf{A}(\lambda, x)$ be the process of running A on input (λ, x) with access to uniformly random coins and assigning the result to y. We usually omit to mention the λ-input and to make the random coins r explicit, we write $\mathsf{A}(x; r)$. With $\mathcal{A}^{\mathsf{B}}$ we denote that $\mathcal{A}$ has oracle access to B. We use $\mathcal{O}$ to denote oracles defined in games and use ϵ to indicate a negligible function. For a positive integer N, we denote the set $\{1, \ldots, N\}$ by $[N]$ and use $\boldsymbol{v} = (v_1, \ldots, v_n)$ to denote a vector. Given two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$, we write $(\boldsymbol{v}, \boldsymbol{w})$ for appending $\boldsymbol{w}$ to $\boldsymbol{v}$. Whenever we have vectors $\boldsymbol{w}$ and $\boldsymbol{v}$ of identical dimension whose components are sets, then by $\boldsymbol{v} \subseteq \boldsymbol{w}$ we mean that the relation is applied componentwise.

2.1 Set Commitments

Fuchsbauer et al. in [FHS19] introduced the notion of a set commitment SC with subset openings. SC allows committing to a set $S \subset Z_p$ by committing to a monic polynomial whose roots are the elements of S and supports openings for sets $T \subseteq S$. We defer the abstract definition to Appendix A.1 and then recall their construction below. We thereby make one property that is satisfied by the set commitment construction in [FHS19] explicit. Namely, we require that the randomness space forms a group and the existence of an algorithm RndmzC such that set commitments and opening information can be perfectly randomized. More formally, we require that for all $\mathsf{pp}_{\mathsf{sc}}$ and $S \in S_{\mathsf{pp}_{\mathsf{sc}}}$ as well as randomness ρ and μ we have that:

$$\mathsf{SC.RndmzC}(\mathsf{SC.Commit}(S; \rho); \mu) = \mathsf{SC.Commit}(S; \rho\mu)$$

Set Commitment Construction of [FHS19]. To simplify our description, we ignore the case that a set S contains the trapdoor α. For a non-empty set S, [FHS19] defines the polynomials $f_S(X) := \prod_{s \in S}(X - s) = \sum_{i=0}^{|S|} f_i \cdot X^i$. Note that for P, since $P^{f_S(\alpha)} = \prod_{i=0}^{|S|} P^{(f_i \cdot \alpha^i)}$, one can efficiently compute $P^{f_S(\alpha)}$ when given $\left(P^{\alpha^i}\right)_{i=0}^{|S|}$:

- $\mathsf{SC.Setup}(1^\lambda, 1^t) \to s\mathsf{pp}_{\mathsf{sc}}$: On input a security parameter λ and a maximum set cardinality t, run $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$, pick $\alpha \leftarrow Z_p$ and output $\mathsf{pp}_{\mathsf{sc}} \leftarrow (\mathsf{BG}, (P^{\alpha^i}, \hat{P}^{\alpha^i})_{i \in [t]})$, which defines message space $S_{\mathsf{pp}_{\mathsf{sc}}} = \{S \subset Z_p | 0 < |S| \leq t\}$. $\mathsf{pp}_{\mathsf{sc}}$ will be an implicit input to all algorithms.
- $\mathsf{SC.Commit}(S) \to (C, O)$: On input a set $S \in S_{\mathsf{pp}_{\mathsf{sc}}}$: pick $\rho \leftarrow Z_p^*$, compute $C \leftarrow (P^{f_S(\alpha)})^\rho \in \mathbb{G}_1^*$ and output (C, O) with $O \leftarrow \rho$.

$\mathsf{SC.Open}(C, S, O) \to 0/1$: On input a commitment C, a set S, and an opening information $O = \rho$: if $C \notin \mathbb{G}_1^*$ or $\rho \notin Z_p^*$ or $S \notin S_{\mathsf{pp_{SC}}}$ then return $\perp$. Otherwise if $O = \rho$ and $C = (P^{f_S(\alpha)})^\rho$, return 1; else return 0.

$\mathsf{SC.OpenSubset}(C, S, O, T) \to W$: On input a commitment C, a set S, an opening information O and a set T, if $0 \leftarrow \mathsf{SC.Open}(C, S, O)$ or $T \not\subseteq S$ or $T = \emptyset$ then return $\perp$. If $O = \rho$, output $W \leftarrow (P^{f_{S \setminus T}(\alpha)})^\rho$.

$\mathsf{SC.VerifySubset}(C, T, W) \to 0/1$: On input a commitment C, a set T and a witness W: if $C \notin \mathbb{G}_1^*$ or $T \notin S_{\mathsf{pp_{SC}}}$, return 0. Else if $W \in \mathbb{G}_1^*$ and $e(W, \hat{P}^{f_T(\alpha)}) = e(C, \hat{P})$, return 1; else return 0.

$\mathsf{SC.RndmzC}(C, O, \mu) \to (C', O')$: On input a set commitment C, an opening information O and a randomness $\mu \in Z_p$, output $C' = C^\mu$ and $O' = \mu \cdot O$.

On computing commitments. We note that with the knowledge of the trapdoor α, we can compute a commitment when externally provided with the randomness ρ in the group as P^ρ. If required, we will therefore modify the commitment computation to $\mathsf{SC.Commit}(S, \alpha, P^\rho)$, which then computes $C \leftarrow (P^\rho)^{f_S(\alpha)}$ and sets $O \leftarrow \perp$.[9]

3 SPSEQ on Updatable Commitments

As our primary building block, we introduce equivalence-class signatures on updatable commitments called (SPSEQ-UC). It can be viewed as a variant of SPSEQ with the following modifications: 1) It considers the message space as vectors of randomizable set commitments, i.e., one can adapt a signature on a commitment vector to a randomized version of the signed commitments. This means that equivalence classes are defined on vectors of commitments. 2) SPSEQ-UC not only considers signing representatives of classes of a single projective equivalence relation $\mathcal{R}$, but a family of relations as we allow to extend signed vectors by additional commitments. Thus we consider a family of such relations $\mathbb{R}^\ell$ such that $\mathcal{R}^k \in \mathbb{R}^\ell$ for any $1 \leq k \leq \ell$. More precisely, in SPSEQ-UC signing of a commitment vector of length k also produces an update key $\mathsf{uk}_{k'}$ corresponding to an integer k' with $k \leq k' \leq \ell$. Given the update key $\mathsf{uk}_{k'}$, in addition to adapt a signature on a commitment vector $\boldsymbol{C}$ in class $[\boldsymbol{C}]_{\mathcal{R}^k}$ to another representative of the given class, one also can update a commitment vector $\boldsymbol{C}$ (i.e., extending it) to a vector $\boldsymbol{C'}$ being in a class $[\boldsymbol{C'}]_{\mathcal{R}^{k'}}$ of a new equivalence relation. Then one can adapt the signature accordingly to the updated commitment vector. 3) In a SPSEQ-UC, a signature is bound to a user public key. The signer produces a signature bound to a user public key, but this signature can be adapted into another valid signature for a new user public key by anyone knowing the old user secret key.

3.1 Formal Definitions

We recall that in an SPSEQ scheme, one can sign vectors of group elements and it is possible to jointly randomize messages and signatures in public. The messages space consists of representatives of projective equivalence classes defined on one source group of a bilinear group, i.e, $\mathbb{G}_1^\ell$ (for some fixed $\ell > 1$ and prime-order group $\mathbb{G}_1$), and randomization of a message represents a change to another representative in the signed class. In case of SPSEQ-UC the message space consists of a vector of group elements of set commitments (a commitment vector) from $(\mathbb{G}_1^*)^\ell$. In contrast to SPS-EQ, we require updating, i.e., extending, the commitment vector and thus we consider a family of equivalence relations $\mathbb{R}^\ell$ so that for any k with $1 < k \leq \ell$, we can define the following equivalence

[9] Here we assume that ρ is honestly chosen, i.e., the discrete logarithm w.r.t. P is known. This can be enforced by requiring to provide a ZKPoK of the discrete logarithm ρ w.r.t. element P.

relation $\mathcal{R}^k \in \mathbb{R}^\ell$ and the equivalence class $[\boldsymbol{C}]_{\mathcal{R}^k}$ of a set commitment vector $\boldsymbol{C} = (C_1, \ldots, C_k)$. More concretely, for a fixed bilinear group BG and (k, ℓ), we define $\mathcal{R}^k \in \mathbb{R}^\ell$ as follows:

$$\mathcal{R}^k = \left\{ (\boldsymbol{C}, \boldsymbol{C}') \in (\mathbb{G}_1^*)^k \times (\mathbb{G}_1^*)^k \Leftrightarrow \exists \mu \in Z_p^* : \boldsymbol{C}' = \boldsymbol{C}^\mu \right\}.$$

Definition 1 (SPSEQ-UC **scheme).** *A* SPSEQ-UC *scheme for a set commitment scheme* SC *and a parameterized family of equivalence relations* $\mathbb{R}^\ell$ *consists of the following PPT algorithms:*

- $\mathsf{PPGen}(1^\lambda, 1^t, 1^\ell) \rightarrow (\mathsf{pp})$: On input the security parameter λ and an upper bound t for the cardinality of committed sets and a length parameter $\ell > 1$, this probabilistic algorithm outputs the public parameters pp. The message set space $S_{\mathsf{pp}_{\mathsf{SC}}}$ is well-defined from pp. pp will be an implicit input to all algorithms.
- $\mathsf{KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{vk}, \mathsf{sk})$: On input the public parameters pp, this probabilistic algorithm outputs a verification and signing key pair $(\mathsf{vk}, \mathsf{sk})$.
- $\mathsf{UKeyGen}(\mathsf{pp}) \rightarrow (\mathsf{sk}_u, \mathsf{pk}_u)$: On input the public parameters pp, this probabilistic algorithm outputs a key pair $(\mathsf{sk}_u, \mathsf{pk}_u)$ for a user u.
- $\mathsf{RndmzC}(\boldsymbol{C}, \boldsymbol{O}, \mu) \rightarrow (\boldsymbol{C}', \boldsymbol{O}')$: This deterministic algorithm takes as input a commitment vector $\boldsymbol{C}$ of size $1 < k \leq \ell$, corresponding openings $\boldsymbol{O}$ and randomness μ. It runs $(C_i', O_I') \leftarrow \mathsf{SC.RndmzC}(C_i, O_i, \mu)$ for all $i \in [k]$ and outputs a new representative of the set commitment vector $\boldsymbol{C}' \in [\boldsymbol{C}]_{\mathcal{R}^k}$ and corresponding openings $\boldsymbol{O}'$.
- $\mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}, k', \mathsf{pk}_u; \boldsymbol{\rho}) \rightarrow (\sigma, (\boldsymbol{C}, \boldsymbol{O}), \mathsf{uk}_{k'})$: This probabilistic algorithm takes as input a signing key sk, a vector of set messages $\boldsymbol{M} = (M_1 \ldots, M_k)$, an index k' with $k \leq k' \leq \ell$, a user public key pk_u and a vector of randomness $\boldsymbol{\rho}$. It computes $(C_j, O_j)_{j \in [k]} \leftarrow \mathsf{SC.Commit}(M_j; \rho_j)$ for all $j \in [k]$, sets $\boldsymbol{C} = (C_1 \ldots, C_k)$ and $\boldsymbol{O} = (O_1, \ldots, O_k)$. It outputs a signature $(\sigma, \boldsymbol{C})$ for pk_u, and also an update key $\mathsf{uk}_{k'}$ in case $k' \neq \ell$.
- $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \boldsymbol{C}, \sigma, (\boldsymbol{T}, \boldsymbol{U})) \rightarrow 0/1$: On input a verification key vk, a user public key pk_u, a commitment vector $\boldsymbol{C} = (C_1, \ldots, C_k)$, the purported signature σ, and a pair $(\boldsymbol{T}, \boldsymbol{U})$, it outputs 0 if any of the following checks fail and 1 otherwise:
  - Check whether σ is a valid signature for $(\boldsymbol{C}, \mathsf{pk}_u)$.
  - For all $(T_i, U_i) \in (\boldsymbol{T}, \boldsymbol{U})$: if $U_i = W_i$ check $1 \stackrel{?}{=} \mathsf{SC.VerifySubset}(C_i, T_i, W_i)$. Else if $U_i = O_i$, check $1 \stackrel{?}{=} \mathsf{SC.Open}(C_i, T_i, O_i)$.
- $\mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) \rightarrow 0/1$: On input a verification key vk, an update key $\mathsf{uk}_{k'}$, an integer k' and a signature σ, this deterministic update key verification algorithm outputs 0 or 1.
- $\mathsf{RndmzPK}(\mathsf{pk}_u, \psi, \chi) \rightarrow \mathsf{pk}_u'$: On input a user public key pk_u and randomness ψ, χ, this public key randomization algorithm outputs the randomized public key pk_u'.
- $\mathsf{ChangeRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\boldsymbol{C}, \boldsymbol{O}), \sigma, \mu, \psi) \rightarrow (\sigma', (\boldsymbol{C}', \boldsymbol{O}'), (\mathsf{uk}_{k'}' \text{ or } \perp), \mathsf{pk}_u', \chi)$: This algorithm takes as input the user public key pk_u, a commitment vector $\boldsymbol{C} = (C_1, \ldots, C_k)$ in equivalence class $[\boldsymbol{C}]_{\mathcal{R}^k}$ and corresponding openings $\boldsymbol{O}$, a signature σ for $\boldsymbol{C}$, randomness ψ, μ and optionally an update key $\mathsf{uk}_{k'}$. It returns an updated signature σ' for a new commitment vector and corresponding openings $(\boldsymbol{C}', \boldsymbol{O}) \leftarrow \mathsf{RndmzC}(\boldsymbol{C}, \boldsymbol{O}, \mu)$ such that $\boldsymbol{C}' \in [\boldsymbol{C}]_{\mathcal{R}^k}$ as well as a randomized user public key $\mathsf{pk}_u' \leftarrow \mathsf{RndmzPK}(\mathsf{pk}_u, \psi, \chi)$ for uniform randomness χ. In case that $\mathsf{uk}_{k'} \neq \perp$, it additionally outputs a randomized update key $\mathsf{uk}_{k'}'$.
- $\mathsf{ChangeRel}(M_l, \sigma, \boldsymbol{C}, \mathsf{uk}_{k'}, k'') \rightarrow (\sigma', (\boldsymbol{C}', O_l), \mathsf{uk}_{k''})$: On input a message set $M_l \subset S_{\mathsf{pp}_{\mathsf{SC}}}$ for $l = k + 1 \in [k']$, a signature σ for a vector of commitments representative $\boldsymbol{C} = (C_1, \ldots, C_k)$ of equivalence class $[\boldsymbol{C}]_{\mathcal{R}^k}$, an updatable key $\mathsf{uk}_{k'}$, and an index $k'' \leq k'$. This algorithm adapts a signature σ' for a new commitment vector $\boldsymbol{C}' = (\boldsymbol{C}, C_l)$ of equivalence class $[\boldsymbol{C}']_{\mathcal{R}^l}$, where C_l is a set commitment for M_l with the related opening information O_l. Also, for $k'' \in [l+1, k']$, updates the updatable key for the range $[l+1, k'']$ into $\mathsf{uk}_{k''}$.

$\mathsf{SendConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \sigma) \to (\sigma_{\text{orph}})$: It is an algorithm run by a user who wants to delegate a signature σ. It takes as input the public verification key vk, a secret key sk_u and the signature σ. It outputs an orphan signature σ_{orph}.

$\mathsf{ReceiveConvertSig}(\mathsf{vk}, \mathsf{sk}_{u'}, \sigma_{\text{orph}}) \to \sigma'$: It is an algorithm run by a user who receives a delegatable signature. It takes as input the verification key vk, a secret key $\mathsf{sk}_{u'}$, an orphan signature σ_{orph}. It outputs a new signature σ' for $\mathsf{pk}_{u'}$.

For simplicity, when we write $\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma)$ we mean $[\mathsf{SendConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \sigma) \leftrightarrow \mathsf{ReceiveConvertSig}(\mathsf{vk}, \mathsf{sk}_{u'})] \to \sigma'$, where σ' is a valid signature.

3.2 Security Definitions

Similar to a standard signature scheme, a SPSEQ-UC scheme should also be correct and unforgeable. And similar to SPSEQ we need some additional properties regarding the distribution of modified signatures.

Correctness. As usual we require that honest signatures verify as expected. Moreover, each of the algorithms $\mathsf{ConvertSig}, \mathsf{ChangeRep}$ and $\mathsf{ChangeRep}$ outputs valid signatures for the respective parameters. We provide a formal correctness definition in Appendix C.1.

Unforgeability. For unforgeability, we consider an adversary that has access to signatures for message set vectors of its choice, controls randomness of commitments and is allowed to create as well as corrupt user keys. We require that it cannot come up with a signature on a commitment vector that opens to non-signed message (sub-)sets where we need to consider that the adversary is allowed to extend commitment vectors. In addition, the adversary needs to specify the used user secret and public key $(\mathsf{sk}^*, \mathsf{pk}^*)$ for the forgery, which we require to make it useful in the application to DAC. Note that since the output of $\mathsf{ChangeRel}$ and $\mathsf{ChangeRep}$ is distributed identical to Sign, we do not require to provide access to such oracles as it can be done by the adversary on its own. To detect that signatures derived with $\mathsf{uk}_{k'}$ are obtained from $\mathsf{ChangeRel}$ or are generated freshly in the Sign oracle, we define the following relation $\mathcal{R}_{k'}$. Consequently, signatures that can be legally derived using $\mathsf{ConvertSig}$ and $\mathsf{ChangeRep}$ are not considered as forgeries.

Definition 2. *Let k, ℓ be integers. For any $\ell \geq k' > k$, we define the relation $\mathcal{R}_{k'}$ for two vectors $\boldsymbol{M} = (M_1, \ldots, M_k)$ and $\boldsymbol{M}^* = (M_1^*, \ldots, M_{k'}^*)$ as follows:*

$$(\boldsymbol{M}, \boldsymbol{M}^*) \in \mathcal{R}_{k'} \iff \forall i \leq k : M_i^* \subseteq M_i$$

We formally define the unforgeability game as follows:

Definition 3 (Unforgeability). *A SPSEQ-UC scheme is unforgeable if, for all $(\lambda, t) \in \mathbb{N}$, and $\ell > 1$, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\lambda)$ such that $\Pr[\mathsf{ExpUnf}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda, \ell, t) = 1] \leq \epsilon(\lambda)$, where the experiment $\mathsf{ExpUnf}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda, \ell, t)$ is defined in Fig 1 and Q is the set of queries that $\mathcal{A}$ has issued to the signing oracle.*

Note that unforgeability allows the adversary to either output a full opening ($U_i^* = O_i$) or subset opening ($U_i^* = W_i$) for each commitment in the commitment vector. This is also used to make it useful in the application to DAC.

Privacy notions. Subsequently, we define privacy properties that are similar in vein to origin-hiding or signature adaption in previous works [CL19, FHS19]. However, since SPSEQ-UC supports more functionality for the sake of reduced complexity we present three notions. Firstly with *origin-hiding* we want to guarantee that fresh and randomized signatures are indistinguishable. Secondly,

$\underline{\mathsf{ExpUnf}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda,\ell,t)\text{:}}$

- $Q := \emptyset; \mathcal{UL} := \emptyset, \mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$;
- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$;
- $((\mathsf{sk}^*_u, \mathsf{pk}^*_u)(\boldsymbol{C}^*, \boldsymbol{T}^*, \boldsymbol{U}^*), \sigma^*) \leftarrow \mathcal{A}^{<\mathcal{O}>}(\mathsf{vk}, \mathsf{pp})$

return:

$$\begin{pmatrix} \forall(\mathsf{pk}_u, \mathsf{sk}_u) \notin \mathcal{UL}, \forall(\boldsymbol{M}, k', \mathsf{pk}_u) \in Q : \\ (\boldsymbol{M}, \boldsymbol{T}^*) \notin \mathcal{R}_{k'} \wedge (\mathsf{sk}^*, \mathsf{pk}^*) \in \mathsf{UKeyGen}(\mathsf{pp}) \\ \wedge\ \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}^*_u, \boldsymbol{C}^*, \sigma^*, (\boldsymbol{T}^*, \boldsymbol{U}^*)) = 1 \end{pmatrix}$$

$\underline{\mathcal{O}^{\mathsf{Sign}}(\boldsymbol{M}, k', \mathsf{pk}_u, \boldsymbol{\rho})\text{:}}$

- If $\ell \geq k' \geq k$:
- Then $(\sigma, (\boldsymbol{C}, \boldsymbol{O}), \mathsf{uk}_{k'}) \leftarrow \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}, k', \mathsf{pk}_u; \boldsymbol{\rho})$
- $Q = Q \cup \{(\boldsymbol{M}, k', \mathsf{pk}_u)\}$,
- **return** $((\boldsymbol{C}, \boldsymbol{O}), \sigma, \mathsf{uk}_{k'})$
- Else **return** $\perp$

$\underline{\mathcal{O}^{\mathsf{Create}}(i)\text{:}}$

- $(\mathsf{pk}_u, \mathsf{sk}_u) \leftarrow \mathsf{KeyGen}()$
- $\mathcal{UL} \leftarrow \mathcal{UL} \cup \{(i, \mathsf{pk}_u, \mathsf{sk}_u)\}$
  return (pk_u)

$\underline{\mathcal{O}^{\mathsf{Corrupt}}(i)\text{:}}$

- If $\exists i \in \mathcal{UL}$ such that $(\mathsf{pk}_u, \mathsf{sk}_u) \in \mathcal{UL}$
- Then delete the item from the list and **return** $(\mathsf{sk}_u, \mathsf{pk}_u)$
- Else **return** $\perp$

Fig. 1. Experiment $\mathsf{ExpUnf}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda, \ell, t)$.

with *derivation-privacy* we want to guarantee that signatures for extended commitment vectors are indistinguishable from fresh ones. Thirdly, with *conversion-privacy* we want to guarantee that when switching a user key in the signature, the resulting signature is indistinguishable from a fresh signature. Subsequently, we use $\approx$ to denote perfect indistinguishability.

Origin-hiding (also called signature adaptation [FHS15, FHS19]) formalizes the fact that signatures for well-formed commitments and well-formed update keys the output of ChangeRep is distributed identical to fresh signatures on the new representative.

Definition 4 (Origin-hiding). *For all* (λ, t, ℓ) *and* $\mathsf{pp} \in \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$*, for all* vk*,* pk_u*,* $\boldsymbol{C}$*,* $\boldsymbol{M}$*,* $\boldsymbol{O}$*,* $\boldsymbol{T}$*,* $\boldsymbol{U}$*,* $\mathsf{uk}_{k'}$*,* k' *and* σ*. If* $\mathsf{SC.Open}(\mathsf{pp}, C_j, M_j, O_j)_{j \in k} = 1 \ \wedge\ \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \boldsymbol{C}, \sigma, (\boldsymbol{T}, \boldsymbol{U})) = 1$*, then for all* μ, ψ*, the algorithm* $\mathsf{ChangeRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\boldsymbol{C}, \boldsymbol{O}), \sigma, \mu, \psi)$ *outputs a uniformly random* $\boldsymbol{C}' \in [\boldsymbol{C}]_{\mathcal{R}^k}$ *and uniformly random* pk'_u*, and* σ' *in the respective spaces.*

Since we support the extension of the signed commitment vector, with derivation privacy we guarantee that signatures derived on a message vector $\boldsymbol{C}^*$ output by ChangeRel are indistinguishable from signatures freshly created with sk by Sign on the extended vector.

Definition 5 (Derivation-privacy). *For all* (λ, t, ℓ)*,* $\mathsf{pp} \in \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$*, all* $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$*,* pk_u*,* $\boldsymbol{M}$*,* $\boldsymbol{O} = \boldsymbol{\rho}$*,* $\boldsymbol{T}$*,* $\boldsymbol{U}$*,* $\mathsf{uk}_{k'}$*,* k'*, and* σ*. If* $\mathsf{SC.Open}(\mathsf{pp}, C_j, M_j, O_j)_{j \in k} = 1 \wedge \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \boldsymbol{C}, \sigma, (\boldsymbol{T}, \boldsymbol{U})) = 1 \wedge \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$*, then, for all* $k'' \in [k{+}1, k']$*,* M_l*, we have* $(\sigma', (\boldsymbol{C}', O_l), \mathsf{uk}_{k''}) \leftarrow \mathsf{ChangeRel}(M_l, \sigma, \boldsymbol{C}, \mathsf{uk}_{k'}, k'')$ *and the following holds:*

$$(\mathsf{vk}, \mathsf{sk}, \mathsf{pk}_u, \mathsf{uk}_{k'}, (\sigma', (\boldsymbol{C}', \boldsymbol{O}'), \mathsf{uk}_{k''})) \approx$$
$$(\mathsf{vk}, \mathsf{sk}, \mathsf{pk}_u, \mathsf{uk}_{k'}, \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}', k'', \mathsf{pk}_u; \boldsymbol{\rho}))$$

where $\boldsymbol{M}' = (\boldsymbol{M}, M_l)$ *and* $\boldsymbol{O}' = (\boldsymbol{O}, O_l)$.

With conversion-privacy we require that a converted signature, i.e., a signature where the public key has been switched, is identically distributed to a fresh signature using the new public key.

Definition 6 (Conversion-privacy). *For all* (λ, t, ℓ), $\mathsf{pp} \in \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$, *and for all* $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$, $(\mathsf{sk}_u, \mathsf{pk}_u) \in \mathsf{UKeyGen}$, $\boldsymbol{C}, \boldsymbol{T}, \boldsymbol{M}, \boldsymbol{U}, \boldsymbol{O} = \boldsymbol{\rho}$, $\mathsf{uk}_{k'}$, k', *and* σ. *If* $\mathsf{SC.Open}(\mathsf{pp}, C_j, M_j, O_j)_{j\in k} = 1 \ \wedge \ \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \boldsymbol{C}, \sigma, (\boldsymbol{T}, \boldsymbol{U})) = 1 \ \wedge \ \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$, *then for all* $(\mathsf{pk}_{u'}, \mathsf{sk}_{u'}) \in \mathsf{UKeyGen}$, *the following holds:*

$$(\mathsf{vk}, \mathsf{sk}, \mathsf{pk}_{u'}, (\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma), (\boldsymbol{C}, \boldsymbol{O}), \mathsf{uk}_{k'})) \approx$$
$$(\mathsf{vk}, \mathsf{sk}, \mathsf{pk}_{u'}, \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}, k', \mathsf{pk}_{u'}; \boldsymbol{\rho})).$$

Remark 1. We can also define a class-hiding notion in the vein of [CL19, FHS19]. For completeness we include it in Appendix C.2, but analogous to what is shown in [FHS15] (Proposition 1), the origin-hiding notion together with the indistinguishability of the message space (which holds under the DDH assumption), we achieve a stronger notion. We will use the above properties in combination with the DDH assumption later directly in the proof of anonymity of the DAC.

3.3 Construction

We now present our SPSEQ-UC construction and start with an intuition behind our approach. We start from the SPSEQ signature scheme in [FHS15]. Inspired by their implicit use of SC in [FHS19] to construct traditional ACs, we make the message space of the scheme a vector of set commitments in a way that meets our requirements. Consequently, SPSEQ-UC encodes each message set $M_i \subset Z_p^t$ into a set commitment C_i and signs a commitment vector of size $k \leq \ell$ being a vector in $(\mathbb{G}_1^*)^k$. The randomization of the set commitment vector is identical to a change of representative in the SPSEQ. More specifically, we use the algorithm SC.Commit to encode a message set to a set commitment in Sign and also an algorithm RndmzC to change a set commitment representative. Note that as made explicit in the unforgeability game, we allow the adversary to control the randomness (opening information) used in commitments. So we choose this randomness externally and pass it to Sign which then passes it to SC.Commit. The most significant change compared to SPSEQ is the update of the commitment vector (appending more set commitments to the vector), and the support to adapt a signature to this updated commitment vector. We can use the elements from the set commitment parameters $\{P^{a^i}\}_{0<i<t}$ and bind them to the signing key of the respective position of the vector and the randomness used in the Sign. We do this for all indices from k to k' and finally use these elements as the update key. Now, one can insert a new commitment (message set) to the signature by another algorithm ChangeRel that receives a signature with new message sets and the update key. It first encodes this set to the set commitment. Then, it uses the update key to create another set commitment value for the new message set, which can be easily aggregated into the signature (element Z) and get the new signature for new message sets. For the user's public key bound to a signature, instead of singing the user's public key like the commitment vector, we define the extra element T to tie the signature to the user's public key. This allows us to update the user's public keys by only locally updating the T element in the signature but still guaranteeing unforgeability. The RndmzPK then allows to randomize a public key consistently with the signature and in a way to achieve a new independent user public key for each call to ChangeRep.

$\mathsf{PPGen}(1^\lambda, 1^t, 1^\ell) \to (\mathsf{pp})$: Run $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$. Pick $\alpha \leftarrow Z_p^*$ and run $\mathsf{pp}_{\mathsf{SC}} = (P^{\alpha^i}, \hat{P}^{\alpha^i})_{i\in[t]} \leftarrow \mathsf{SC.Setup}(1^\lambda, 1^t; \alpha)$, and define $S_{\mathsf{pp}_{\mathsf{SC}}} \leftarrow \{M \subset Z_p | 0 < |M| \leq t\}$. Output $\mathsf{pp} = \{\mathsf{BG}, \mathsf{pp}_{\mathsf{SC}}, S_{\mathsf{pp}_{\mathsf{SC}}}, I\}$.

$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{vk}, \mathsf{sk})$: For $0 \leq i \leq \ell$ pick $x_i \leftarrow (Z_p^*)^\ell$, set the signing key $\mathsf{sk} = (x_0, \ldots, x_\ell)$. Compute the related verification key $\mathsf{vk} = (X_0, \hat{X}_0, \ldots, \hat{X}_\ell)$, where $X_0 = P^{x_0}$ and $\hat{X}_i = \hat{P}^{x_i}$ for $0 \leq i \leq \ell$. Output $(\mathsf{vk}, \mathsf{sk})$.

$\mathsf{UKeyGen}(\mathsf{pp}) \to (\mathsf{sk}_u, \mathsf{pk}_u)$: Pick $w_u \leftarrow Z_p^*$, set $\mathsf{pk}_u \leftarrow P^{w_u}$ and $\mathsf{sk}_u = w_u$, and finally return $(\mathsf{sk}_u, \mathsf{pk}_u)$.

$\mathsf{RndmzC}(\boldsymbol{C}, \boldsymbol{O}, \mu) \to \boldsymbol{C}'$: On input a set commitment vector $\boldsymbol{C} \in [\boldsymbol{C}]_{\mathcal{R}^k}$ corresponding openings $\boldsymbol{O}$ and randomness μ, produces a new representative of the set commitment vector $\boldsymbol{C}' = \boldsymbol{C}^\mu$ and corresponding openings $\boldsymbol{O}' = \mu\boldsymbol{O}$.

$\mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}, k', \mathsf{pk}_u; \boldsymbol{\rho}) \to (\sigma, (\boldsymbol{C}, \boldsymbol{O}), \mathsf{uk}_{k'})$: On input the signing key sk, a vector of message sets $\boldsymbol{M} = (M_1 \ldots, M_k)$, an index k', $k \leq k' \leq \ell$, a user public key pk_u and a vector of randomness $\boldsymbol{\rho}$. For all $j \in [k]$ run $(C_j, O_j)_{j\in[k]} \leftarrow \mathsf{SC.Commit}(M_j; \rho_j)$, and get a vector of set commitments $\boldsymbol{C} = (C_1 \ldots, C_k)$ related to a vector of sets $\boldsymbol{M}$ and opening information $\boldsymbol{O} = (O_1, \ldots, O_k)$. More precisely, $\mathsf{SC.Commit}$ computes a set commitment for each M_j in $\boldsymbol{M}$ as follows: define a polynomial $f_{M_j}(X) := \prod_{m\in M_j}(X - m) = \sum_{i=0}^{|M_j|} f_i \cdot X^i$ and with $\rho_j \in \boldsymbol{\rho}$, compute:

$$C_j = \left(\prod_{i=0}^{|M_j|} \left(P^{\alpha^i}\right)^{f_i}\right)^{\rho_j} \text{ and } O_j = \rho_j,$$

where P^{α^i} are group elements in pp. Then, compute a signature σ for $(\mathsf{pk}_u, \boldsymbol{C})$ as follows: Pick a random $y \leftarrow Z_p^*$ and compute $\sigma =$

$$\left(Z \leftarrow \left(\prod_{j\in[k]} C_j^{x_j}\right)^{\frac{1}{y}}, Y \leftarrow P^y, \hat{Y} \leftarrow \hat{P}^y, T \leftarrow P^{x_1 \cdot y} \cdot \mathsf{pk}_u^{x_0}\right)$$

Also, if $k \neq \ell$, compute an update key for a range between k and k' as:

$$\mathsf{uk}_{k'} = \left(\left(\mathsf{usign}_j = \left((P^{\alpha^i})^{x_j}\right)^{y^{-1}}\right)_{j\in[k+1,k'],i\in[t]}\right).$$

Output $(\sigma, (\boldsymbol{C}, \boldsymbol{O}), \mathsf{uk}_{k'})$.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \boldsymbol{C}, \sigma, (\boldsymbol{T}, \boldsymbol{U})) \to 0/1$: On input a verification key vk, a user public key pk_u, a commitment vector $\boldsymbol{C} = (C_1, \ldots, C_k)$, the purported signature σ, and a pair $(\boldsymbol{T}, \boldsymbol{U})$, it outputs 0 if any of the following checks fail and 1 otherwise:

- Check whether σ is a valid for $(\boldsymbol{C}, \mathsf{pk}_u)$, i.e., output 0 if one of the following checks fails:

$$\prod_{j=1}^{k} e(C_j, \hat{X}_{j+1}) = e(Z, \hat{Y}) \quad \wedge \quad e(Y, \hat{P}) = e(P, \hat{Y})$$
$$\wedge \quad e(T, \hat{P}) = e(Y, \hat{X}_1) \cdot e(\mathsf{pk}_u, \hat{X}_0).$$

- For all $(T_i, U_i) \in (\boldsymbol{T}, \boldsymbol{U})$ if $U_i = W_i$, then W_i is a witness for T_i being subset of the set committed to C_i: run $\mathsf{SC.VerifySubset}(C_i, T_i, W_i)$, i.e., output 1 if the following equation holds; else 0:

$$\prod_{i\in[|\boldsymbol{W}|]} e(W_i, \hat{P}^{f_{T_i}(\alpha)}) = \prod_{i\in[|\boldsymbol{W}|]} e(C_i, \hat{P})$$

Else $U_i = O_i$, then $T_i = M_i$ is a message set and O_i is a valid opening of C_i to T_i: run $\mathsf{SC.Open}(C_i, T_i, O_i)$, i.e., output 1 if the following holds; else 0:

$$\forall i \in [k] : O_i = \rho_i \wedge C_i = (P^{f_{M_i}(\alpha)})^{\rho_i}$$

$\mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) \to 0/1$. On input a vk, $\mathsf{uk}_{k'}$, index k', a signature $\sigma = (Z, Y, \hat{Y}, T)$ output 1 if the following holds; else 0:

$$\prod_{j\in[k']} \prod_{i\in[t]} e(P^{\alpha^i}, \hat{X}_j) = \prod_{j\in[k']} e(\mathsf{usign}_j, \hat{Y})$$

$\mathsf{RndmzPK}(\mathsf{pk}_u, \psi, \chi) \to \mathsf{pk}'$: On input a user public key pk_u and randomness $\psi, \chi \in Z_p^*$, output the randomized public key $\mathsf{pk}'_u = (\mathsf{pk}_u \cdot P^{\chi})^{\psi}$, related to the secret key $(\chi + \mathsf{sk}_u)\psi$.

$\mathsf{ChangeRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\boldsymbol{C}, \boldsymbol{O}), \sigma, \mu, \psi) \to (\sigma', (\boldsymbol{C}', \boldsymbol{O}'), (\mathsf{uk}'_{k'} \text{ or } \perp), \mathsf{pk}'_u, \chi)$: On input a user public key pk_u, optionally an update key $\mathsf{uk}_{k'}$, a commitment vector $\boldsymbol{C} = (C_1, \ldots, C_k)$ in equivalence class $[\boldsymbol{C}]_{\mathcal{R}^k}$ with corresponding openings $\boldsymbol{O}$, a valid signature σ for $\boldsymbol{C}$ and randomness $\psi, \mu \in Z_p^*$. Pick $\chi \leftarrow Z_p^*$ and compute a new commitment representative $(\boldsymbol{C}', \boldsymbol{O}') \leftarrow \mathsf{RndmzC}(\boldsymbol{C}, \boldsymbol{O}, \mu)$ as well as a randomized user public key $\mathsf{pk}'_u \leftarrow \mathsf{RndmzPK}(\mathsf{pk}_u, \psi, \chi)$ and update signature as $\sigma' = (Z^{\frac{\mu}{\psi}}, Y^{\psi}, \hat{Y}^{\psi}, (T \cdot X_0^{\chi})^{\psi})$. Moreover, if $\mathsf{uk}_{k'} \neq \perp$, check if $\mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$, then randomize the update key as

$$\mathsf{uk}'_{k'} = \left((\mathsf{usign}_j^{\mu \cdot \psi^{-1}})_{j\in[k+1,k']} \right),$$

and output $(\sigma', (C', \boldsymbol{O}'), (\mathsf{uk}'_{k'} \text{ or } \perp), \mathsf{pk}'_u, \chi)$.

$\mathsf{ChangeRel}(M_l, \sigma, \boldsymbol{C}, \mathsf{uk}_{k'}, k'') \to (\sigma', (\boldsymbol{C}', O_l), \mathsf{uk}_{k''})$: On input a message set $M_l \subset S_{\mathsf{pp_{SC}}}$ for $l = k + 1 \in [k']$, a valid signature σ for commitment vector $\boldsymbol{C} = (C_1, \ldots, C_k)$ in equivalence class $[\boldsymbol{C}]_{\mathcal{R}^k}$, a valid update key $\mathsf{uk}_{k'}$, and an index $k'' \leq k'$. First it creates a set commitment $(C_l, O_l = \rho_l) \leftarrow \mathsf{SC.Commit}(M_l)$. Then, it performs the following steps to update the signature for a commitment vector including C_l:

- First, compute a set commitment ϑ_l as in $\mathsf{SC.Commit}$, but using keys of the l-component of $\mathsf{uk}_{k'}$ as

$$\mathsf{usign}_l = \left((P^{\alpha^i})^{x_l} \right)^{y^{-1}} \quad \text{for } i \in [t] :$$

$$f_{M_l}(X) = \sum f_i X^i \Rightarrow \vartheta_l = (\prod_{i\in[t]} \mathsf{usign}_{li}^{f_i})^{\rho_l} = \prod_{i\in[t]} ((\underbrace{P^{\alpha^i \cdot x_l \cdot y^{-1}}}_{\mathsf{usign}_{li}})^{f_i})^{\rho_l}$$

- Second, update σ for a commitment vector $\boldsymbol{C}' = (\boldsymbol{C}, C_l)$ as follows:

$$\sigma' = \left((Z \cdot \vartheta_l), Y, \hat{Y}, T \right).$$

- Finally for $k'' \in [l + 1, k']$, update the update key $\mathsf{uk}_{k'}$ for $j \in [l + 1, k'']$:

$$\mathsf{uk}_{k''} = \left(\mathsf{usign}_j \right)_{j\in[l+1,k'']}.$$

Output $(\sigma', (\boldsymbol{C}', O_l), \mathsf{uk}_{k''})$.

$\mathsf{SendConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \sigma) \to \sigma_{\mathrm{orph}}$: On input $\mathsf{vk} = (X_0, \hat{X}_0, \ldots, \hat{X}_\ell)$, a user secret key sk_u for the public key pk_u, and a valid signature $\sigma = (Z, Y, \hat{Y}, T)$. Output an orphan signature $\sigma_{\mathrm{orph}} =$

$$\left(Z, Y, \hat{Y}, T' = T \cdot (X_0^{\mathsf{sk}_u})^{-1} \right)$$

$\mathsf{ReceiveConvertSig}(\mathsf{vk}, \mathsf{sk}_{u'}, \sigma_{\text{orph}}) \to \sigma'$: On input the verification key vk, a secret key $\mathsf{sk}_{u'}$, an orphan signature $\sigma_{\text{orph}} = \left(Z, Y, \hat{Y}, T'\right)$. Output a signature σ' for $\mathsf{pk}_{u'}$ as:

$$\sigma' = \left(Z, Y, \hat{Y}, T'' = T' \cdot X_0^{\mathsf{sk}_{u'}} = P^{x_1 \cdot y} \cdot \mathsf{pk}_{u'}^{x_0}\right).$$

The correctness of our SPSEQ-UC construction follows form inspection. We formally show the following:

Theorem 1 (Unforgeability). *Our* SPSEQ-UC *construction is unforgeable in the generic group model for type-3 bilinear groups.*

The proof of Theorem 1 is provided in Appendix D.1.

Theorem 2 (Privacy). *Our* SPSEQ-UC *construction is origin-hiding, conversion-privacy and derivation Privacy based on definitions 4, 5 and 6, respectively.*

The proof of Theorem 2 is provided in Appendix D.2.

3.4 Cross-Set Commitment Aggregation

In this section, we introduce an aggregatable set commitment CSCA that allows non-interactive aggregation of witnesses across multiple commitments. We use a technique inspired by [BDFG20, GRWZ20] to batch different subset opening witnesses into one and to improve the efficiency of the verification operation with batching pairing equations. Functionality-wise, we require that witnesses for multiple subsets of multiple commitments can be aggregated into a single value called proof. This allows us to use the CSCA in the DAC scheme in order to open any subset of attributes in each set commitment efficiently. CSCA adds two additional algorithms in SC to aggregate witnesses across k-commitments and verify them:

- $\mathsf{AggregateAcross}(\{C_j, T_j, W_j\}_{j\in[k]}) \to \pi$. Takes as input a collection $\{C_j, T_j\}_{j\in[k]}$ along with the corresponding subset opening witnesses $\{W_j\}_{j\in[k]}$ (computed using $\mathsf{OpenSubset}$) and outputs an aggregated proof π.
- $\mathsf{VerifyAcross}(\{C_j, T_j\}_{j\in[k]}, \pi) \to b$. Takes as input a collection $(\{C_j, T_j\}_{j\in[k]})$ along with a cross-commitment-aggregated proof π, and checks: for all $j \in [k]$, C_j is a set commitment to a message set consistent with the subset T_j.

On computing commitments. Similar to SC in Section 2.1, with the knowledge of the trapdoor α, we can compute a commitment when externally provided with the randomness ρ in the group as P^ρ. So, we add the commitment computation to $\mathsf{CSCA.Commit}_2(S_j, \alpha, P^\rho)$ with addition input to commit group elements.

Construction A Cross Set commitment aggregation scheme CSCA consists of the following PPT algorithms:

- $\mathsf{CSCA.Setup}(1^\lambda, 1^t) \to \mathsf{pp}_{\mathsf{CSCA}}$: On input a security parameter λ and a maximum set cardinality t, run $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$, choose $H : \{0,1\}^* \to Z_p$, pick $\alpha \leftarrow Z_p$, store α as a trapdoor and output $\mathsf{pp}_{\mathsf{CSCA}} \leftarrow (\mathsf{BG}, H, (P^{\alpha^i}, \hat{P}^{\alpha^i})_{i\in[t]})$, which defines message space $S_{\mathsf{pp}_{\mathsf{CSCA}}} = \{S \subset Z_p | 0 < |S| \leq t\}$. $\mathsf{pp}_{\mathsf{CSCA}}$ will be an implicit input to all algorithms.
- $\mathsf{CSCA.Commit}(S_j) \to (C_j, O_j)$: On input a set $S_j \in S_{\mathsf{pp}_{\mathsf{SC}}}$: pick $\rho_j \leftarrow Z_p$, compute $C_j \leftarrow (P^{f_{S_j}(\alpha)})^{\rho_j} \in \mathbb{G}_1^*$ and output (C_j, O_j) with $O_j \leftarrow \rho_j$.

$\mathsf{CSCA.Commit}_2(S_j, \alpha, P^{\rho_j}) \to (C_j, O_j)$: On input a set $S_j \in S_{\mathsf{pp_{SC}}}$, α, and P^{ρ_j}: compute $C_j \leftarrow (P^{\rho_j})^{f_{S_j}(\alpha)} \in \mathbb{G}_1^*$ and output (C_j, O_j) with $O_j \leftarrow \perp$.

$\mathsf{CSCA.Open}(C_j, S_j, O_j) \to 0/1$: On input a commitment C_j, a set S_j, and opening $O_j = \rho_j$: if $C_j \notin \mathbb{G}_1^*$ or $\rho_j \notin Z_p^*$ or $S_j \notin S_{\mathsf{pp_{CSCA}}}$ then return $\perp$. Otherwise if $O_j = \rho_j$ and $C_j = (P^{f_{S_j}(\alpha)})^{\rho_j}$, return 1; else return 0

$\mathsf{CSCA.OpenSubset}(C_j, S_j, O_j, T_j) \to W_j$: On input a commitment C_j, a set S_j, opening O_j and a subset T_j, if $\mathsf{CSCA.Open}(C_j, S_j, O_j)$ or $T_j \not\subseteq S_j$ or $T_j = \emptyset$ then return $\perp$. If $O_j = \rho_j$, output $W_j \leftarrow (P^{f_{S_j \setminus T_j}(\alpha)})^{\rho_j}$.

$\mathsf{CSCA.VerifySubset}(C_j, T_j, W_j) \to 0/1$: On input the commitment C_j, the subset T_j and the witness W_j: if $C_j \notin \mathbb{G}_1^*$ or $T_j \notin S_{\mathsf{pp_{SC}}}$, return 0. Else if $W_j \in \mathbb{G}_1^*$ and $e(W_j, \hat{P}^{f_{T_j}(\alpha)}) = e(C_j, \hat{P})$, return 1; else 0.

$\mathsf{CSCA.AggregateAcross}(\{C_j, T_j, W_j\}_{j\in[k]}) \to \pi$. Takes as input a collection $(\{C_j, T_j\}_{j\in[k]})$ along with the corresponding subset opening witnesses $\{W_j\}_{j\in[k]}$ and outputs an aggregated proof π as follows:

$$\pi := \prod_{j\in[k]} W_j^{t_j}, \text{ where } t_j = H(j, \{C_j, T_j\}_{j\in[k]}).$$

$\mathsf{CSCA.VerifyAcross}(\{C_j, T_j\}_{j\in[k]}, \pi) \to 0/1$. Checks that the following equation holds:

$$\prod_{j\in[k]} e(C_j, \hat{P}^{t_j \cdot Z_{S\setminus T_j}(\alpha)}) = e(\pi, \hat{P}^{Z_S(\alpha)})$$

where $S = \bigcup_j T_j$, and $Z_S(\alpha) = \prod_{i\in S}(\alpha - i)$.

We require the same correctness of opening as before, extended to cross-commitment aggregation in a natural way. To fit with the DAC, one provides one group element in $\mathbb{G}_1$ proof π of the current values of the attributes required for the showing. When multiple subsets of disclosed attribute sets are used, cross-commitment aggregation allows us to compress witnesses $(W_1, \ldots, W_k)$ into a single proof π. This can help to reduce the bandwidth overhead significantly and improves verification efficiency by saving k pairing equations. Note that without aggregation verification has the form: $\prod_{i\in[k]} e(W_i, \hat{P}^{f_{T_i}(\alpha)}) = \prod_{i\in[k]} e(C_i, \hat{P})$.

Randomization. We can randomize π and commitments C_j as $(\pi^\mu, \boldsymbol{C}^\mu)$ and verification works out.

Security. We can view the set commitment from Section 2.1 used for the cross-commitment aggregation as an instantiation of [BDFG20], but restricted to monic polynomials. So their analysis carries over. Note that in AggregateAcross, to be non-interactive, we aggregate witnesses using t_j using a hash function H modeled as a random oracle (as done in [GRWZ20]), meaning that their analysis carries over.

4 Delegatable Anonymous Credentials

We now define our delegatable anonymous credentials DAC application. Our definition is similar to [CL19, FHS19] but splits up the issuing protocol for issuing a root credential (CreateCred) and a delegatable credential (IssueCred). Our model works as follows: A root issuer (called CA) issues a level-0 credential (a root credential) to intermediate issuers using the CreateCred protocol. The credential is assigned for a user $\mathsf{sk_u}$ (whom it knows with a pseudonym $\mathsf{nym_u}$) with an attribute

vector A and the related set commitments $\boldsymbol{C}$ rooted at $\mathsf{pk}_{\mathsf{CA}}$. CA also creates a delegation key $\mathsf{dk}_{k'}$ which determines how the credential can be delegated any further regarding an index k'. With this key, a user U can replace an old pseudonym with a new one and also delegate their credential further to another user, say R, by signing/updating the R's public key and (possibly another) set of attributes A'. More precisely, to delegate a credential, the issuer U (user) needs to derive a signature on the receiver's sk_{R} without being given sk_{R} itself by signing/updating the receiver's pseudonym $\mathsf{nym}_{\mathsf{R}}$ and removing her pseudonym $\mathsf{nym}_{\mathsf{u}}$ and the corresponding sk_{u}. Showing the credential consists of the user proving possession of the secret key sk_{u} associated with his pseudonym and a signature with attributes fulfilling subsets D.

Definition 7 (Delegatable anonymous credentials). DAC *includes algorithms* (Setup, KeyGen, NymGen) *and protocols* CreateCred/ReceiveCred, IssueCred/ReceiveCred *for issuing a credential and* CredProve/CredVerify *for possession of a credential as:*

$\mathsf{Setup}(1^\lambda, 1^t, 1^\ell) \to (\mathsf{pp}, \mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}})$: Takes as input the security parameters λ, an upper bound t for the cardinality of committed sets and a length parameter $\ell > 1$. It generates the public parameters pp for the system as well as a signing key $\mathsf{sk}_{\mathsf{CA}}$ and public key $\mathsf{pk}_{\mathsf{CA}}$ for all $i \in [\ell]$ for CA. Outputs the pp and CA key pair $(\mathsf{pp}, \mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}})$. pp will be an implicitly input to all algorithms.

$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$: Generates a key pair $(\mathsf{pk}, \mathsf{sk})$, where sk is referred to the user's secret key, while pk is a public key (or an initial pseudonym).

$\mathsf{NymGen}(\mathsf{pk}) \to (\mathsf{nym}, \mathsf{aux})$: Takes as input a user's public key pk, and outputs a pseudonym nym for this user and the auxiliary information aux needed to use nym.

Issuing a root credential:

$[\mathsf{CreateCred}(k', \mathsf{A}, \mathsf{sk}_{\mathsf{CA}}) \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_u, \mathsf{A})] \to (\mathsf{cred}, (\boldsymbol{C}, \boldsymbol{O}), \mathsf{dk}_{k'})$: This is an interactive protocol between a user (issuer) who is known by $\mathsf{nym}_{\mathsf{u}}$ and CA. The common inputs are the pp, the CA 's public key $\mathsf{pk}_{\mathsf{CA}}$ and the arbitrate vector A. CA creates a root credential, i.e. the (powerful) delegatable credential for a set commitment vector $\boldsymbol{C}$ corresponding to the attributes vector $\mathsf{A} = (A_1, \ldots, A_k)$ and the related opening information $\boldsymbol{O}$ as well as a delegatable key $\mathsf{dk}_{k'}$ regarding the index k' for a user $\mathsf{nym}_{\mathsf{u}}$ ($\mathsf{nym}_{\mathsf{u}}$ is sent to the CA), rooted at $\mathsf{pk}_{\mathsf{CA}}$.

Issuing/delegating a credential:

$[\mathsf{IssueCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{k'}, \mathsf{sk}_{\mathsf{u}}, \mathsf{cred}_u, A_l, k'') \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{R}}, A_l)] \to (\mathsf{cred}_{\mathsf{R}}, \mathsf{dk}'_{k'})$: It is an interactive protocol between an issuer who is known by $\mathsf{nym}_{\mathsf{u}}$ and runs the IssueCred algorithm, and a receiver, who is known by $\mathsf{nym}_{\mathsf{R}}$ and runs the ReceiveCred side. The common inputs are the pp, the CA 's public key $\mathsf{pk}_{\mathsf{CA}}$, and the attributes set A_l. Also, the issuer takes as input the issuer's secret key sk_{u} and his own credential cred_u (a signature) together with all information associated with it (e.g. A, the delegatable key $\mathsf{dk}_{k'}$, the pseudonym $\mathsf{nym}_{\mathsf{u}}$ and its associated auxiliary information aux_u), and (optionally) an index $k'' < k'$. The receiver takes as input her own secret key sk_{R}, and creates a $\mathsf{nym}_{\mathsf{R}}$ (and sends to $\mathsf{nym}_{\mathsf{u}}$), and the auxiliary information $\mathsf{aux}_{\mathsf{R}}$ associated with her pseudonym $\mathsf{nym}_{\mathsf{R}}$. At the end of the protocol, the receiver side outputs her credential $\mathsf{cred}_{\mathsf{R}}$ that is issued for $\mathsf{A}' = (\mathsf{A}, A_l)$ and $\mathsf{dk}'_{k'}$ (if the delegation is intended to happen more) or $\perp$.

Proof of possession of a credential:

$[\mathsf{CredProve}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{nym}_{\mathsf{P}}, \mathsf{aux}_{\mathsf{P}}, \mathsf{cred}_{\mathsf{P}}, D) \leftrightarrow \mathsf{CredVerify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{nym}_{\mathsf{P}}, D)] \to (0, 1)$: It is an interactive protocol among a prover, who proves possession of a credential and runs the CredProve

side of the protocol, and a verifier, who runs the $\mathsf{CredVerify}$ side. The common inputs are the pp, the CA 's public key $\mathsf{pk}_{\mathsf{CA}}$, and subsets D over attributes sets. The prover takes as input his user secret sk_{P} and his credential together with all information associated with it (i.e., A, the prover's pseudonym $\mathsf{nym}_{\mathsf{P}}$ and corresponding auxiliary information $\mathsf{aux}_{\mathsf{P}}$). The verifier takes common inputs and receives $\mathsf{nym}_{\mathsf{P}}$, output 1 if it accepts the proof of possession of a credential for D and 0 otherwise.

Note that the nym's can be derived from the respective secret key and algorithms (KeyGen, NymGen), we avoid passing nym's as an explicit input whenever possible.

Definition 8 (Correctness of DAC). DAC *is correct if* Setup, KeyGen, *and* NymGen *are run correctly and also the* CreateCred *and* issuing/receiving *protocols are executed correctly on honestly generated inputs, the receiver outputs a credential* $\mathsf{cred}_{\mathsf{R}}$*, when used as input to the prover in an honest execution of the* proving/verifying *protocol, is accepted by the verifier with probability 1.*

4.1 Security of DAC

We define our security model based on the game-based framework in [FHS19], with some modifications to harmonize their definition with the one on SPSEQ-UC. The adversary $\mathcal{A}$ has access to oracles that describe the possible ways to interact with the system. We use $\langle\mathcal{O}\rangle$ to denote the collection of oracles in the games. For the anonymity game we have $\langle\mathcal{O}\rangle = (\mathcal{O}^{\mathsf{User}}, \mathcal{O}^{\mathsf{Corrupt}}, \mathcal{O}^{\mathsf{CreateRoot}}, \mathcal{O}^{\mathsf{ObtIss}}, \mathcal{O}^{\mathsf{Obtain}}, \mathcal{O}^{\mathsf{RootObt}}, \mathcal{O}^{\mathsf{CredProve}})$ and for the unforgeability game $\langle\mathcal{O}\rangle = (\mathcal{O}^{\mathsf{User}}, \mathcal{O}^{\mathsf{CreateRoot}}, \mathcal{O}^{\mathsf{Corrupt}}, \mathcal{O}^{\mathsf{ObtIss}}, \mathcal{O}^{\mathsf{RootIss}}, \mathcal{O}^{\mathsf{Issue}}, \mathcal{O}^{\mathsf{CredProve}})$.

We define four global lists that are shared among oracles as $\mathcal{HU}$ a list of honest users, $\mathcal{CU}$ a list of corrupted users, $\mathcal{L}_{uk}$ a list of user's keys, and $\mathcal{L}_{\mathsf{cred}}$ a list of user-credential pairs which includes issued credentials and corresponding attributes and to which user they were issued. Note that $\mathsf{dk}_{k'}$ implicitly shows k' and subsequently $\mathcal{R}_{k'}$. Moreover, in each issuing query ($\mathcal{O}^{\mathsf{ObtIss}}, \mathcal{O}^{\mathsf{Issue}}$) only one commitment (and attributes set) is added in the commitment vector. Also, for simplicity, we assume that cred_i contains $(\sigma, (\boldsymbol{C}, \boldsymbol{O}, \mathsf{pk}_i = \mathsf{nym}_i), \mathsf{aux}_i)$.

- $\mathcal{O}^{\mathsf{User}}(i)$: Takes as input a user identity i. If $i \in \mathcal{HU}$ or $i \in \mathcal{CU}$ it returns $\perp$, else it creates a fresh entry i in lists $\mathcal{HU}$ and $\mathcal{L}_{uk}$ by running $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and adding i and $(\mathsf{sk}_i, \mathsf{pk}_i)$ to the list $\mathcal{HU}$ and $\mathcal{L}_{uk}$, receptively. It returns $\mathsf{pk}_i = \mathsf{nym}_i$.
- $\mathcal{O}^{\mathsf{Corrupt}}(i, \mathsf{pk}_i)$: Takes as input a user identity i and (optionally) a user public key pk_i. If $i \notin \mathcal{HU}$, a new corrupt user with public key pk_i (or nym_i) is registered and add $i \in \mathcal{CU}$, else it moves the entry corresponding to i from the list of honest users $\mathcal{HU}$ and adds it to the list of corrupted users $\mathcal{CU}$. Then, it returns sk_i and all the associated credentials items $(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i)$ of $\mathcal{L}_{\mathsf{cred}}[i]$. Finally, it sets the form $(\perp, \mathsf{A}, k', \perp) \in \mathcal{L}_{\mathsf{cred}}$ for all A and i in this case.
- $\mathcal{O}^{\mathsf{CreateRoot}}(i, k', \mathsf{A})$: Takes as input a user identity i, an index k' and attributes A. If $i \notin \mathcal{HU}$ it returns $\perp$, else it creates a root credential by running

$$\begin{bmatrix}\mathsf{CreateCred}(k', \mathsf{A}, \mathsf{sk}_{\mathsf{CA}}) \leftrightarrow \\ \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_i, \mathsf{A})\end{bmatrix} \rightarrow (\mathsf{cred}_i, \mathsf{dk}_{k'})$$

  with a user i for an attribute set A and appends $(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$.
- $\mathcal{O}^{\mathsf{RootIss}}(k', \mathsf{A})$: Takes as input an index k' and attributes A. It creates a root credential by running the $\mathsf{CreateCred}$ protocol with $\mathcal{A}$: $\mathsf{CreateCred}(k', \mathsf{A}, \mathsf{sk}_{\mathsf{CA}}) \leftrightarrow \mathcal{A}$ for an attribute set A and appends $(\perp, \mathsf{A}, k', \perp)$ to $\mathcal{L}_{\mathsf{cred}}$. This oracle allows an adversary represented by nym_i to play a corrupted user to get a root credential from a CA.

$\mathcal{O}^{\mathsf{RootObt}}(i, k', \mathsf{A})$: On input a user identity i, an index k' and attributes A. If $i \notin \mathcal{HU}$ it returns $\perp$, else it creates a root credential by running the Receive protocol with $\mathcal{A}$ who impersonates a malicious CA to issue a root credential to an honest user i by running: $\mathcal{A} \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_i, \mathsf{A})$. If $\mathsf{cred}_i = \perp$ the oracle returns $\perp$. Else it stores the resulting $(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i) \in \mathcal{L}_{\mathsf{cred}}$.

$\mathcal{O}^{\mathsf{ObtIss}}(i, j, A_l, k'')$: Takes as user identities i and j, a set of attributes A_l, and (optionally) an index k''. It makes user i delegate a credential to user j. If $i, j \notin \mathcal{HU}$ or $(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i) \notin \mathcal{L}_{\mathsf{cred}}$ it returns $\perp$, else finds entries $(\mathsf{sk}_i, \mathsf{cred}_i)$, sk_j, and runs the issuing protocols as:

$$\begin{bmatrix} \mathsf{IssueCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{k'}, \mathsf{sk}_i, \mathsf{cred}_i, A_l, k'') \\ \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_j, A_l) \end{bmatrix} \rightarrow (\mathsf{cred}_j, \mathsf{dk}'_{k'})$$

and adds the entry $(j, \mathsf{A}', \mathsf{dk}'_{k'}, \mathsf{cred}_j)$ to $\mathcal{L}_{\mathsf{cred}}$, where $\mathsf{A}' = (\mathsf{A}, A_l)$.

$\mathcal{O}^{\mathsf{Obtain}}(j, A_l, k'')$: On input a user identity $j \in \mathcal{HU}$ and a set of attributes A_l, and (optionally) an index k''. If $j \notin \mathcal{HU}$ it returns $\perp$. Else, the oracle runs the Receive protocol with $\mathcal{A}$: $\mathcal{A} \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_j, A_l)$. If $\mathsf{cred}_j = \perp$ the oracle returns $\perp$. Else it stores the resulting output $(\mathsf{cred}_j, \mathsf{dk}'_{k'}, \mathsf{A}')$ and it appends $(j, \mathsf{A}', \mathsf{dk}'_{k'}, \mathsf{cred}_j)$ to $\mathcal{L}_{\mathsf{cred}}$. This oracle is used by $\mathcal{A}$, whom it knows by nym_i impersonating an issuer to issue a credential to an honest user j.

$\mathcal{O}^{\mathsf{Issue}}(i, A_l, k'')$: On input a user identity i, a set of attributes A_l, and (optionally) an index k''. If $i \notin \mathcal{HU}$ it returns $\perp$. Also it checks if $\nexists(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i) \in \mathcal{L}_{\mathsf{cred}}$, returns $\perp$. Else, it runs: $\mathsf{IssueCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{k'}, \mathsf{sk}_i, \mathsf{cred}_i, A_l, k'') \leftrightarrow \mathcal{A}$. The elements $(\perp, \mathsf{A}' = (\mathsf{A}, A_l), k', \perp)$ are then added to $\mathcal{L}_{\mathsf{cred}}$. This oracle is used by a corrupted user with adversarial nym_j to get a credential from honest issuer i.

$\mathcal{O}^{\mathsf{CredProve}}(j, D)$: On input an index of an issuance j and subsets D. This oracle first parses $\mathcal{L}_{\mathsf{cred}}[j]$ as $(i, \mathsf{A}', \mathsf{dk}'_{k'}, \mathsf{cred}_i)$. Let cred_i be the credential issued on A' for a user i during the i-th query to $\mathcal{O}^{\mathsf{ObtIss}}$ or $\mathcal{O}^{\mathsf{Obtain}}$ (or it can be outputs of $\mathcal{O}^{\mathsf{CreateRoot}}$ when directly issued by CA). If $i \notin \mathcal{HU}$ returns $\perp$. Else, it retrieves $(\mathsf{aux}_i, \mathsf{nym}_i, \mathsf{sk}_i)$ for i from lists cred_i and $\mathcal{L}_{uk}$, and runs: $\mathsf{CredProve}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_i, \mathsf{nym}_i, \mathsf{aux}_i, \mathsf{cred}_i, D) \leftrightarrow \mathcal{A}$, with the adversary playing the role of the verifier.

$\underline{\mathsf{ExpAno}^b_{\mathsf{DAC},\mathcal{A}}(\lambda, \ell, t)}$:

- $(\mathsf{pp}, \mathsf{pk}_{\mathsf{CA}}, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, 1^\ell, 1^t)$
- $b' \leftarrow \mathcal{A}^{\langle \mathcal{O}_b^{\mathsf{Anon}}, \mathcal{O} \rangle}(\mathsf{st})$
- **return** $(b = b')$

$\underline{\mathcal{O}_b^{\mathsf{Anon}}(\mathsf{j}_0, \mathsf{j}_1, D)}$:

- If j_0 or $\mathsf{j}_1 > |\mathcal{L}_{\mathsf{cred}}|$ the oracle returns $\perp$.
- Else, it parses $\mathcal{L}_{\mathsf{cred}}[\mathsf{j}_0]$ as $(i_0, \mathsf{A}'_0, \mathsf{dk}_0, \mathsf{cred}_0)$ and $\mathcal{L}_{\mathsf{cred}}[\mathsf{j}_1]$ as $(i_1, \mathsf{A}'_1, \mathsf{dk}_1, \mathsf{cred}_1)$.
- If $D(\mathsf{A}'_0) \neq D(\mathsf{A}'_1) \vee |\boldsymbol{C}_0| \neq |\boldsymbol{C}_1| \vee \mathsf{cred}_b \nleftarrow \mathcal{O}^{\mathsf{ObtIss}}$, **return** $\perp$.
- Otherwise run: $\mathcal{A} \leftrightarrow \mathsf{CredProve}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_b, \mathsf{nym}_b, \mathsf{aux}_b, \mathsf{cred}_b, D)$

$\underline{\mathsf{ExpUnf}_{\mathsf{DAC},\mathcal{A}}(\lambda, \ell, t)}$:

- $(\mathsf{pp}, (\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}})) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^t)$
- $(D^*, \mathsf{nym}^*) \leftarrow \mathcal{A}^{\langle \mathcal{O} \rangle}(\mathsf{pp}, \mathsf{pk}_{\mathsf{CA}})$
- $b \leftarrow (\mathcal{A} \leftrightarrow \mathsf{CredVerify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{nym}^*, D^*))$
- $\forall(\perp, \mathsf{A}', k', \perp) \in \mathcal{L}_{\mathsf{cred}}$: If $(D^*, \mathsf{A}') \in \mathcal{R}_{k'}$, **return** 0
- Else **return** b

Fig. 2. Experiments $\mathsf{ExpAno}_{\mathsf{DAC},\mathcal{A}}(\lambda, \ell, t)$ and $\mathsf{ExpUnf}_{\mathsf{DAC},\mathcal{A}}(\lambda, \ell, t)$.

Anonymity. Anonymity requires that a malicious verifier cannot distinguish between any two users. The adversary has adaptive access to an oracle that on the input of two distinct user indexes i_0 and i_1, acts as one of the two credential owners (depending on bit b) in the verification algorithm. Note that $D(\mathsf{A}') = 1$ if attributes in A' satisfies the policy subset and $D(\mathsf{A}') = 0$ otherwise. Moreover, $\mathsf{cred}_b \leftarrow \mathcal{O}^{\mathsf{ObtIss}}$ requires that credentials cred_0 and cred_1 are on a delegation path from a (corrupted) root credential where all delegations have been performed honestly, but the respective users might all be corrupted. We note that this requirement is similar to the anonymity model of CL in [CL19], however, we additionally allow the adversary to access the user corruption oracle in which we reveal the user's secret keys to the adversary. CL can not support this type of corruption as then the anonymity of their construction breaks down.This makes our model stronger than the one of CL. The essence of the game is captured by the oracles $\mathcal{O}_b^{\mathsf{Anon}}$ in Fig 2. To make the game non-trivial, we impose restrictions that the policy is either satisfied or not by both credentials and they have commitment vectors of equal length. Note this also implies unlinkability for delegation anonymity.

Definition 9 (Anonymity). *A* DAC *is anonymous, if for all* $(\lambda, \ell, t) \in \mathbb{N}$*, any* PPT *adversary* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\lambda)$ *so that* $|\Pr[\mathsf{ExpAno}^0_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t) = 1] - \Pr[\mathsf{ExpAno}^1_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t) = 1]| \leq \frac{1}{2} + \epsilon(\lambda)$*, experiments are defined in Fig 2, respectively.*

Unforgeability. Unforgeability requires that no adversary can convince a verifier into accepting a credential for a set of attributes for which he does not possess credentials. Intuitively, an adversary wins the unforgeability experiment (cf. Fig 2) if he is able to convince an honest verifier that he satisfies a certain policy while does not have an appropriate credential.

Definition 10 (Unforgeability). *A* DAC *is unforgeable if, for all* $(\lambda, \ell, t) \in \mathbb{N}$*, for any* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $\epsilon(\lambda)$ *such that* $\Pr[\mathsf{ExpUnf}_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t) = 1] \leq \epsilon(\lambda)$*, where the experiment* $\mathsf{ExpUnf}_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t)$ *is defined in Fig 2.*

4.2 Construction of DAC

We provide a concrete instantiation of DAC based on SPSEQ-UC signature (which we denote by Σ) defined in Section 3 and CSCA schemes 3.4 (see Fig 3). For a more comprehensive description, we use the following notation: Assume $\mathcal{U} = S_{\mathsf{pp}_{\mathsf{SC}}}$, $\mathsf{A} = \boldsymbol{M}$, the updatable key as a delegatable key $\mathsf{uk}_{k'} = \mathsf{dk}_{k'}$ and the root authority's public key $\mathsf{pk}_{\mathsf{CA}} = \mathsf{vk}$. Then each credential is parameterized with a set $\mathsf{A} = (A_1, \ldots, A_k) \subseteq \mathcal{U}$, and consider the relation in Definition 2 for attributes which defines how a user can use their credentials. For the sake of compactness we write $\mathsf{ZKPoK}(w, x)$ (or $\mathsf{NIZK}(w, x)$) for an (non-interactive) zero-knowledge proof of knowledge of witness w for statement x and $\mathsf{ZKPoK}(w, x) = 1$ if the verifier accepts (see Appendix A.2 for a more formal treatment). In Fig 3 we replace SC by CSCA (cf. Section 3.4) to improve the communication bandwidth by aggregating witnesses and also verification efficiency due to batching of pairing equations. We stress that CSCA is fully compatible with SC and provides identical functionality. Note that in order to get a credential in IssueCred and CreateCred protocols, a user needs to send one of his pseudonyms to the CA (or an issuer respectively) and use some mechanism to authenticate with the real identity (e.g., physically showing a driving license). This is outside the protocol and we omit it here. Regarding the first two commitments in CreateCred protocol, we can assume the first dummy commitment for a fixed set (used by all credentials) and the second commitment for the respective attribute set A. We only require this for the first delegation.

On compactness of credentials. We note that credentials are constant-size as the SPSEQ-UC

signature is constant size. Although a delegation key depends on parameter k, it only applies to intermediate issuers and not to end-users who do not hold such a key. Another aspect is the commitment randomness, which represents auxiliary data. Each commitment randomness represents a scalar that allows recomputing the commitment. So it is sufficient to store the randomness. In the extreme case where the whole commitment vector comes from one party, one can compute all randomness for the commitments via a PRF function and just store a seed.

Theorem 3. *The* DAC *construction in Fig 3 is correct, unforgeable, and anonymous.*

The proof is presented in Appendix D.3, where we note that in our formal proof, we consider our core SPSEQ-UC which is based on SC. Unforgeability follows from the ZKPoK and unforgeability of SPSEQ-UC. While anonymity follows from the ZKPoK, privacy properties of SPSEQ-UC and the DDH assumption. Note that when using CSCA instead of SC and NIZK obtained via the Fiat-Shamir heuristic, then the proofs are in the random oracle model (ROM).

Remark 2. We note that except for the NIZK in Setup, we require multiple-extractions in the security proofs. Thus we opted to rely on interactive ZKPoK. We however note that when willing to pay some extra costs, one could instead use straight-line extractable NIZK, e.g., obtained via Fischlin's transformation [Fis05, Kas22].

5 Implementation and Performance Evaluation

This section presents our evaluation results in terms of computation and communication overhead based on an implementation of the proposed SPSEQ-UC and DAC schemes.

Experimental Results. We implement a Python library for our DAC system and the underlying SPSEQ-UC. Our implementation is based upon the `bplib` library[10] and petlib[11] with `OpenSSL` bindings[12]. We use the popular pairing friendly curve BN256 which provides efficient type 3 bilinear groups at a security level of around 100 bit.

We present a benchmark of SPSEQ-UC (including the cross-commitment aggregation provided by the CSCA scheme) and the DAC scheme described in Section 4. DAC uses Schnorr-style discrete-logarithm zero-knowledge proofs (using Damgard's technique [CDM00] for obtaining malicious-verifier interactive zero-knowledge proofs of knowledge) during the showing and issuing/delegating of credentials. It also uses non-interactive zero-knowledge proofs of knowledge obtained via the Fiat-Shamir heuristic for proofs of knowledge of $\mathsf{pk}_{\mathsf{CA}}$. Our measurements have been performed on an Intel Core i5-6200U CPU at 2.30 GHz, 16 GB RAM running Ubuntu 20.04.3. For our evaluation, we take the execution time of each algorithm for the following parameters: ℓ represents an upper bound for the length of commitment vector, t an upper bound for the cardinality of committed sets, $n_i < t$ stands for the number of attributes in each attribute set A_i in the respective commitment C_i of the commitment vector, which we set to be the same for every level (in each commitment) for simplicity. Moreover, k represents the length of attributes set vector $\mathsf{A} = (A_1, \ldots, A_k)$ and thus commitment vector $\boldsymbol{C}$ and k' is the number of attribute sets which can be delegated.

The results are shown in Table 2, where μ_{AV} is the mean and SD the standard deviation of 100 executions of each algorithm. Delegation assumes that each time one more attribute set is added to the credential. More precisely, in Table 2, we set the above parameters as $t = 25$, $\ell = 15$, $k = 4$, $k' = 7$ and $n = 10$ to cover many different use-cases. ChangeRep/uk represents the randomization

[10] https://github.com/gdanezis/bplib
[11] https://github.com/gdanezis/petlib
[12] https://github.com/dfaranha/OpenPairing

$\mathsf{Setup}(1^\lambda, 1^t, 1^\ell)$: Pick $\alpha \leftarrow Z_p$ and run $\mathsf{pp}_\Sigma \leftarrow \Sigma.\mathsf{Setup}(1^\lambda, 1^t, 1^\ell; \alpha)$. Generate a key pair $(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}})$ for CA as $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{KeyGen}(\mathsf{pp}_\Sigma)$, and set $\mathsf{pk}_{\mathsf{CA}} = (\mathsf{vk}, P^\alpha)$, and $\mathsf{sk}_{\mathsf{CA}} = (\mathsf{sk}, \alpha)$, and run $\mathsf{NIZK}(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}})$. Output $\mathsf{pp} = (\mathsf{pp}_\Sigma, \mathsf{pk}_{\mathsf{CA}}, \mathsf{NIZK}(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}}))$. The attribute space is $\mathcal{U} = S_{\mathsf{pp}_\Sigma}$ of pp_Σ.

$\mathsf{KeyGen}(\mathsf{pp})$: A user u picks $w_u \leftarrow Z_p^*$, sets $\mathsf{pk}_u \leftarrow P^{w_u}$ and $\mathsf{sk}_u = w_u$, and finally returns $(\mathsf{sk}_u, \mathsf{pk}_u)$.

$\mathsf{NymGen}(\mathsf{pk}_u)$: Pick randomness $\psi, \chi \leftarrow Z_P^*$ and compute $\mathsf{nym}_u = \mathsf{pk}'_u \leftarrow \Sigma.\mathsf{RndmzPK}(\mathsf{pk}_u, \psi, \chi)$. Set $\mathsf{aux}_u = (\chi, \psi)$, and output pairs $(\mathsf{nym}_u, \mathsf{aux}_u)$.

Issuing a root credential:

U picks $\rho_j \leftarrow Z_p^*$ for a commitment with attributes set $A_j \in \mathsf{A}$ for $j \in [k]$, with $k \geq 2$, creates nym_u and sends $((P^{\rho_j})_{j\in[k]}, \mathsf{nym}_u)$ to CA.[a] Also, it proves via running an interactive $\mathsf{ZKPoK}(\boldsymbol{\rho}, P^{\boldsymbol{\rho}})$, that it knows randomness (opening information).

CA checks if proofs are correct, then runs $(\sigma, (\boldsymbol{C}, \boldsymbol{O}), \mathsf{uk}_{k'}) \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{A}, k', \mathsf{pk}_u; (P^{\rho_j})_{j\in[k]})$, where $(\boldsymbol{C}, \boldsymbol{O}) \leftarrow \mathsf{CSCA.Commit}_2(A_i, \alpha, P^{\rho_j})_{i\in[k]}$ for the attribute vector $\mathsf{A} = (A_1, \ldots, A_k)$ and $\mathsf{nym}_u = \mathsf{pk}_u$. CA sets $\mathsf{dk}_{k'} = \mathsf{uk}_{k'}$ and outputs $(\sigma, \mathsf{dk}_{k'})$ as well as $(\boldsymbol{C}, \boldsymbol{O})$. Note that $\boldsymbol{O} = \perp$, since opening information have been selected already by U.

Finally, U sets $\boldsymbol{O} = \boldsymbol{\rho}$ and runs $(\sigma', (\boldsymbol{C}', \boldsymbol{O}'), \mathsf{dk}'_{k'}, \mathsf{nym}'_u, \chi) \leftarrow \Sigma.\mathsf{ChangeRep}(\mathsf{nym}_u, \mathsf{dk}_{k'}, (\boldsymbol{C}, \boldsymbol{O}), \sigma, \mu, \psi)$ for μ, ψ and update aux_u with χ, ψ and saves $\mathsf{cred}_u = \sigma'$, as a credential cred as well as $(\mathsf{dk}'_{k'}, \boldsymbol{C}', \boldsymbol{O}')$.

Issuing/delegating a credential (an issuer U and a receiver R):

R creates his pseudonym nym_R and sends it to U.

U parses $(\mathsf{cred}_u = \sigma, \mathsf{dk}_{k'})$ for $(\mathsf{A}, \boldsymbol{C}, \boldsymbol{O})$ and prepares a delegated credential by replacing nym_u with nym_R. It issues the credential regarding an attribute set A_l and nym_R as:

- First U and R run $\sigma' \leftarrow \Sigma.\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma)$ together to switch pseudonyms. Then, to issue $\mathsf{A}' = (\mathsf{A}, A_l)$, U runs $(\sigma'', (\boldsymbol{C}', O_l), \mathsf{uk}_{k''}) \leftarrow \Sigma.\mathsf{ChangeRel}(M_l, \sigma', \boldsymbol{C}, \mathsf{uk}_{k'}, k'')$, where k'' is an index, $\mathsf{uk}_{k'} = \mathsf{dk}_{k'}$ and $M_l = A_l$ for $\boldsymbol{C}' = (\boldsymbol{C}, C_l)$.
- U sends $(\sigma'', \boldsymbol{C}', \boldsymbol{O}' = (\boldsymbol{O}, O_l))$ to R. If delegation is allowed, then it additionally sends $\mathsf{dk}_{k''} = \mathsf{uk}_{k''}$.

R checks $\Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{nym}_\mathsf{R}, \boldsymbol{C}', \sigma'', (\mathsf{A}', \boldsymbol{O}')) = 1$, then for μ, ψ runs $(\sigma''', (\boldsymbol{C}'', \boldsymbol{O}''), \mathsf{uk}'_{k''}, \mathsf{pk}'_u, \chi) \leftarrow \Sigma.\mathsf{ChangeRep}(\mathsf{nym}_\mathsf{R}, \mathsf{dk}_{k''}, (\boldsymbol{C}', \boldsymbol{O}'), \sigma'', \mu, \psi)$ and updates aux_R with χ, ψ and stores the resulting as credentials $(\mathsf{cred}_\mathsf{R} = \sigma''', (\boldsymbol{C}'', \boldsymbol{O}''))$ and the delegatable key $\mathsf{uk}'_{k''} = \mathsf{dk}'_{k''}$.

Proof of possession of a credential (a prover P and a verifier V):

P who already knows a credential like $\sigma = \mathsf{cred}_\mathsf{R}$ for nym_R and $(\boldsymbol{C}, \boldsymbol{O})$ receives subsets $D = \{d_j\}_{j\in[k]}$ and prepares a credential on pseudonym nym_P and $\{A_j\}_{j\in[k]}$:

- Run $(\sigma', (\boldsymbol{C}', \boldsymbol{O}'), \mathsf{pk}'_u, \chi) \leftarrow \Sigma.\mathsf{ChangeRep}(\mathsf{pk}_u, \perp, (\boldsymbol{C}, \boldsymbol{O}), \sigma, \mu, \psi)$ for μ, ψ, where $\sigma = \mathsf{cred}_\mathsf{R}$ and $\mathsf{pk}_u = \mathsf{nym}_\mathsf{R}$. Set $\sigma' = \mathsf{cred}_\mathsf{P}$ and $\mathsf{pk}'_u = \mathsf{nym}_\mathsf{P}$, and update aux_P with ψ, χ.
- Run $W_j \leftarrow \mathsf{CSCA.OpenSubset}(C_j, A_j, O_j, d_j)$ for $j \in [k]$. Aggregate witness $\pi \leftarrow \mathsf{CSCA.AggregateAcross}(\{C_j, d_j, W_j\}_{j\in[k]})$, randomize $\pi' \leftarrow \pi^\mu$.
- P sends $(\mathsf{cred}_\mathsf{P}, \boldsymbol{C}', \mathsf{nym}_\mathsf{P}, \pi')$ to V, and runs an interactive protocol $\mathsf{ZKPoK}(\mathsf{sk}_\mathsf{P}, \mathsf{nym}_\mathsf{P})$ with V which denote zero knowledge proof of $(\mathsf{sk}_\mathsf{P}, \mathsf{aux}_P)$.

V outputs 1, if $\mathsf{ZKPoK}(\mathsf{sk}_\mathsf{P}, \mathsf{nym}_\mathsf{P}) \wedge \Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_u, \boldsymbol{C}, \sigma, (T, \boldsymbol{U}))$ is verified, else outputs 0, where $\sigma = \mathsf{cred}_\mathsf{P}$, $T = D$, $\boldsymbol{U} = \pi'$ and $\mathsf{pk}_u = \mathsf{nym}_\mathsf{P}$.

[a] Note that for issuing a root credential $k = 2$ is sufficient (one can encode all attributes in just two commitments). But for the sake of generality and since it is supported we present it for arbitrary $k \geq 2$.

Fig. 3. Our DAC scheme (Σ denotes SPSEQ-UC)

Table 2. Execution times for SPSEQ-UC and DAC algorithms and protocols in milliseconds

	SPSEQ-UC								DAC			
	Setup	KeyGen	Sign	ChangeRep/uk	ChangeRep	ChangeRel	ConvertSig	Verify	CreateCred	DelegIssue	CredProve	CredVerify
μ_{AV}	385	21	73	50	6	15	2	38	82	21	35	195
SD	±3	±1	±2	±1	±2	±2	±1	±1	±1	±2	±2	±3

of signature along with $\mathsf{uk}_{k'}$ and ChangeRep represents the randomization of signatures only. In the CredProve and CredVerify protocols, we use d_i to denote the subset of each attribute set A_i that will be disclosed. Here, D denotes the set of d_i as $D = (d_i)_{i\in[k]}$. We thereby assume that each subset d_i contains approximately half of the attributes in A_i. Let us provide an example for the showing part to clarify our notation: Assume $k = 2$, we have two commitments C_1, C_2 (for simplicity we can say each commitment is the output of one delegation level e.g., $L = 2$ is the delegation level) such that each includes 5 attributes. Then $D = (d_1, d_2)$ means that each d_1 and d_2 includes two attributes of sets of C_1 and C_2, respectively and $|D| = 4$. In this case, we say that the total number of (messages) attributes is $|\boldsymbol{M}| = |\mathsf{A}| = k \cdot n$ (we also use this interpretation in Figs 4 and 5).

In Fig 4 we show the effect on the computation time of SPSEQ-UC when increasing the parameters (k, k', n). Since the Setup algorithm runs only once, we do not consider the computation time of Setup. We measure the computation time for a message (or an attribute) set of size n from 5 to 20, k from 2 to 10 (so that the total messages $|\boldsymbol{M}|$ is from 10 to 200), and k' from 4 to 20. Since the runtime of the algorithms KeyGen, ChangeRep, and ConvertSig is independent of the parameters (k, k', n) we omit them.

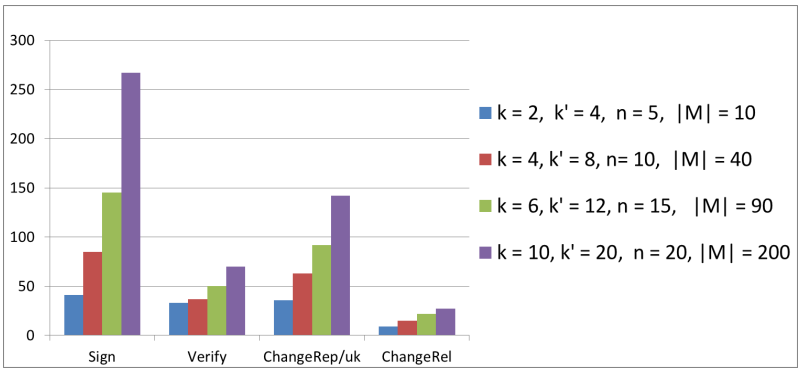
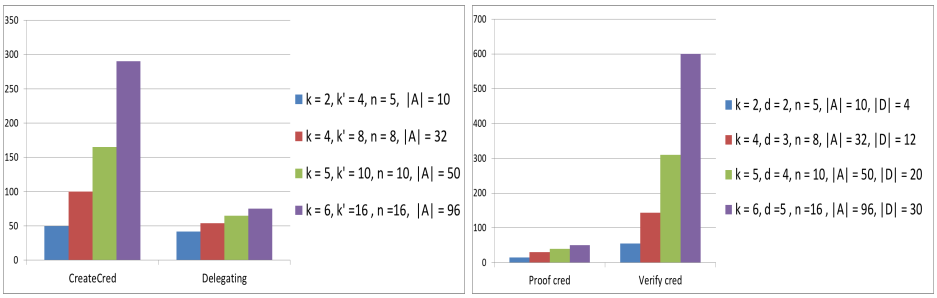


Fig. 4. The running times of SPSEQ-UC (ms)

In Fig 5 we show the effect of increasing the parameters (k, k', d, n) on the computation time of DAC. We measure the computation time of CredProve and CredVerify protocols for n, k and d (a

disclosed attribute subset) varying from 5 to 16, 2 to 6 and 2 to 5, respectively. The total disclosed attributes length $|D| = d \cdot k$ and the total attributes length $|\mathsf{A}| = n \cdot k$ range from 4 to 30 and 10 to 96, respectively. The CredProve and CredVerify are independent of k'. Meanwhile, the computation time of CreateCred and IssueCred change when k' varies from 4 to 16 in addition to being dependent on n and k. These algorithms are independent of d. As can be seen in Fig 5, the pairing product



(a) The running times of IssueCred (b) The running times of ProveCred

Fig. 5. The running times of DAC (ms)

operations in the verification produce the largest overhead. Though, in absolute terms verification is still highly efficient. For example, it is less than a second, in the maximum parameters setting with almost 100 attributes and disclosing 30 attributes. This efficiency makes our implementation suitable for time-critical applications like public transportation, ticketing, etc.

Theoretical Analysis of Performance and Comparison. We analyze the computational and communication complexity compared with Camenisch et al. [CDD17] (CDD), one of the most efficient and fully specified approaches, in Appendix B.

6 Conclusion

In this paper we first present a new primitive called structure-preserving signatures on equivalence classes on updatable commitments SPSEQ-UC in which one can sign vectors of set commitments that can be extended by additional set commitments. Moreover, signatures contain a user's public key, which can be switched. Second, we present an efficient delegatable anonymous credential scheme DAC that supports attributes, provides strong privacy under a reasonable corruption model, and allows the delegators to restrict further delegations. We show the practical efficiency of our DAC by presenting performance benchmarks based on an implementation.

Acknowledgements. This work has in part been carried out within the scope of Digidow, the Christian Doppler Laboratory for Private Digital Authentication in the Physical World and has partially been supported by the LIT Secure and Correct Systems Lab. Omid Mir and René Mayrhofer gratefully acknowledge financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, the Christian Doppler Research Association, 3 Banken IT GmbH, ekey biometric systems GmbH, Kepler Universitätsklinikum GmbH, NXP Semiconductors Austria GmbH and Co KG, Österreichische

Staatsdruckerei GmbH, and the State of Upper Austria. Daniel Slamanig was supported by the European commission through ECSEL Joint Undertaking (JU) under grant agreement n°826610 (COMP4DRONES), the European Union's Horizon 2020 research and innovation programme under grant agreement n°861696 (LABYRINTH) and by the Austrian Science Fund (FWF) and netidee SCIENCE under grant agreement P31621-N38 (PROFET).

References

BB18. Johannes Blömer and Jan Bobolz. Delegatable attribute-based anonymous credentials from dynamically malleable signatures. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 221–239. Springer, Heidelberg, July 2018.

BCC+09. Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 108–125. Springer, Heidelberg, August 2009.

BCET21. Dmytro Bogatov, Angelo De Caro, Kaoutar Elkhiyaoui, and Björn Tackmann. Anonymous transactions with revocation and auditing in hyperledger fabric. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *Cryptology and Network Security - 20th International Conference, CANS 2021, Vienna, Austria, December 13-15, 2021, Proceedings*, volume 13099 of *Lecture Notes in Computer Science*, pages 435–459. Springer, 2021.

BDFG20. Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Report 2020/081, 2020. https://eprint.iacr.org/2020/081.

BEK+21. Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. Issuer-hiding attribute-based credentials. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *CANS 2021*, volume 13099 of *LNCS*, pages 158–178. Springer, 2021.

BF20. Balthazar Bauer and Georg Fuchsbauer. Efficient signatures on randomizable ciphertexts. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 359–381. Springer, Heidelberg, September 2020.

BFPV11. Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 403–422. Springer, Heidelberg, March 2011.

CDD17. Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 683–699. ACM Press, October / November 2017.

CDM00. Ronald Cramer, Ivan Damgård, and Philip D. MacKenzie. Efficient zero-knowledge proofs of knowledge without intractability assumptions. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000*, volume 1751 of *LNCS*, pages 354–372. Springer, Heidelberg, January 2000.

CKLM13. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Succinct malleable NIZKs and an application to compact shuffles. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 100–119. Springer, Heidelberg, March 2013.

CKLM14. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: New definitions and delegatable anonymous credentials. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 199–213. IEEE Computer Society, 2014.

CL03. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, Heidelberg, September 2003.

CL04. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.

CL06. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Heidelberg, August 2006.

CL19. Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 535–555. Springer, Heidelberg, March 2019.

CL21. Elizabeth C. Crites and Anna Lysyanskaya. Mercurial signatures for variable-length messages. *PoPETs*, 2021(4):441–463, October 2021.

CLP22. Aisling Connolly, Pascal Lafourcade, and Octavio Perez-Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022*, volume 13177 of *LNCS*, pages 409–438. Springer, 2022.

CPZ20. Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1445–1459. ACM Press, November 2020.

DGS+18. Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, 2018.

FHS15. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 233–253. Springer, Heidelberg, August 2015.

FHS19. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019.

Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, August 2005.

Fuc11. Georg Fuchsbauer. Commuting signatures and verifiable encryption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 224–245. Springer, Heidelberg, May 2011.

Gro15. Jens Groth. Efficient fully structure-preserving signatures for large messages. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 239–259. Springer, Heidelberg, November / December 2015.

GRWZ20. Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. Pointproofs: Aggregating proofs for multiple vector commitments. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2007–2023. ACM Press, November 2020.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

HS14. Christian Hanser and Daniel Slamanig. Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 491–511. Springer, Heidelberg, December 2014.

HS21. Lucjan Hanzlik and Daniel Slamanig. With a little help from my friends: Constructing practical anonymous credentials. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2004–2023. ACM, 2021.

ILV11. Malika Izabachène, Benoît Libert, and Damien Vergnaud. Block-wise P-signatures and non-interactive anonymous credentials with efficient attributes. In Liqun Chen, editor, *13th IMA International Conference on Cryptography and Coding*, volume 7089 of *LNCS*, pages 431–450. Springer, Heidelberg, December 2011.

Kas22. Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. Cryptology ePrint Archive, Report 2022/393, 2022. https://ia.cr/2022/393.

KLOR20. Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Anonymous tokens with private metadata bit. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 308–336. Springer, Heidelberg, August 2020.

KSD19. Mojtaba Khalili, Daniel Slamanig, and Mohammad Dakhilalian. Structure-preserving signatures on equivalence classes from standard assumptions. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 63–93. Springer, Heidelberg, December 2019.

MSM+18. Sinisa Matetic, Moritz Schneider, Andrew Miller, Ari Juels, and Srdjan Capkun. DelegaTEE: Brokered Delegation Using Trusted Execution Environments. In *27th USENIX Security*, pages 1387–1403, 2018.

PS16. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, February / March 2016.

San20. Olivier Sanders. Efficient redactable signature and application to anonymous credentials. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 628–656. Springer, Heidelberg, May 2020.

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

A Additional Preliminaries

A.1 Set Commitments

Definition 11 (Set commitment [FHS19]). *A set commitment scheme* SC *consists of the following PPT algorithms.*

- $\mathsf{SC.Setup}(1^\lambda, 1^t) \to \mathsf{pp}_{\mathrm{sc}}$: This probabilistic algorithm takes as input a security parameter λ and an upper bound t for the cardinality of committed sets, both in unary form. It outputs public parameters $\mathsf{pp}_{\mathrm{sc}}$ (which include a description of an efficiently samplable message space $S_{\mathsf{pp}_{\mathsf{SC}}}$ containing sets of maximum cardinality t). $\mathsf{pp}_{\mathrm{sc}}$ will be an implicit input to all algorithms.
- $\mathsf{SC.Commit}(S) \to (C, O)$: This probabilistic algorithm takes as input a non-empty set $S \in S_{\mathsf{pp}_{\mathsf{SC}}}$. It outputs a commitment C to set S and opening information O.
- $\mathsf{SC.Open}(C, S, O) \to 0/1$: This deterministic algorithm takes as input a commitment C, a set S and opening information O. If O is a valid opening of C to $S \in S_{\mathsf{pp}_{\mathsf{SC}}}$, it outputs 1, and 0 otherwise.
- $\mathsf{SC.OpenSubset}(C, S, O, T) \to W$: Takes as input a commitment C, a set $S \in S_{\mathsf{pp}_{\mathsf{SC}}}$, opening information O and a nonempty set T. It returns $\perp$ if $T \not\subseteq S$; else it returns a witness W for T being a subset of the set S committed to in C.
- $\mathsf{SC.VerifySubset}(C, T, W) \to 0/1$: This deterministic algorithm takes as input a commitment C, a non-empty set T and a witness W. If W is a witness for T being a subset of the set committed to in C, it outputs 1, and 0 otherwise.

We refer the reader to [FHS19] for a formal definition of the correctness, binding, hiding and subset-soundness notions.

A.2 Zero-Knowledge Proofs of Knowledge

We define zero-knowledge proofs of knowledge (ZKPoK) and discuss non-interactive versions thereof (NIZK). In our DAC, we require protocols to prove knowledge of discrete logarithm relations. This can be efficiently realized by relying on Sigma protocols (i.e., three-round public-coin honest-verifier zero-knowledge proofs of knowledge). Sigma protocols are efficient instantiations of ZKPoK which can be converted to (malicious-verifier) zero-knowledge proofs of knowledge, using Damgard's Technique [CDM00] and made non-interactive using different techniques (discussed below).

ZKPoK. Let $L_{\mathcal{R}} = \{x \mid \exists\, w : (x,w) \in \mathcal{R}\} \subseteq \{0,1\}^* \times \{0,1\}^*$ is a binary, polynomial-time (witness) relation. For such a relation, the membership of $x \in L_{\mathcal{R}}$ can be decided in polynomial time (in $|x|$) when given a witness w of length polynomial in $|x|$ certifying $(x,w) \in \mathcal{R}$. We assume an interactive protocol $(\mathcal{P},\mathcal{V})$ between a prover $\mathcal{P}$ and a PPT verifier $\mathcal{V}$ and denote the outcome of the protocol as $(\cdot, b) \leftarrow \big(\mathcal{P}(\cdot,\cdot), \mathcal{V}(\cdot)\big)$ where $b = 0$ indicates that $\mathcal{V}$ rejects and $b = 1$ that it accepts the conversation with $\mathcal{P}$. We require the following properties:

Definition 12 (Completeness). *We call an interactive protocol $(\mathcal{P},\mathcal{V})$ for a relation $\mathcal{R}$ complete if for all $x \in L_{\mathcal{R}}$ and w such that $(x,w) \in R$ we have that $(\cdot, 1) \leftarrow \big(\mathcal{P}(x,w), \mathcal{V}(x)\big)$ with probability 1.*

Definition 13 (Zero knowledge (ZK)). *An $(\mathcal{P},\mathcal{V})$ for a language L is ZK if for any (malicious) verifier $\mathcal{V}^*$, there exists a PPT algorithm $\mathcal{S}$ (the simulator) such that:*

$$\{\mathcal{S}(x)\}_{x\in L} \approx \{< (\mathcal{P},\mathcal{V}^\star)(x) >\}_{x\in L},$$

where $h(\mathcal{P},\mathcal{V}^*)(x)i$ shows the transcript of the communication between $\mathcal{P}$ and $\mathcal{V}^\star$ on the common input x.

Definition 14 (Knowledge soundness). *We say that $(\mathcal{P},\mathcal{V})$ is a proof of knowledge (PoK) relative to an NP relation $\mathcal{R}$ if for any malicious prover $\mathcal{P}^*$ such that $(\cdot, 1) \leftarrow \big(\mathcal{P}^*(x), \mathcal{V}(x)\big)$ with probability greater than ϵ there exists a PPT knowledge extractor K (with rewinding black-box access to $\mathcal{P}^*$) such that $K^{\mathcal{P}^*}(x)$ returns a value w satisfying $(x,w) \in \mathcal{R}$ with probability polynomial in ϵ.*

If all properties hold, then we denote this interactive protocol as a zero-knowledge proofs of knowledge (ZKPoK).

Non-Interactive Zero-Knowedge Proofs (of Knowledge). One can use the Fiat-Shamir heuristic to transform any Sigma protocol into a non-interactive zero-knowledge proof of knowledge (NIZK). Whenever one requires multiple-extractions in a security proof, a standard measure is to opt for interactive ZKPoK. We however note that when willing to pay some extra costs, one could instead use straight-line extractable NIZK, e.g., obtained via Fischlin's transformation [Fis05, Kas22].

B Theoretical analysis

In this section, we compare the asymptotic efficiency of our DAC with CDD [CDD17].

Table 3. Computational complexity

	CDD [CDD17]	Our DAC
CredProve	**odd:** $\sum_{i=1,3}^{L} 1\mathbb{G}_2 + (n_i+2)\mathbb{G}_1 + (1+d_i)\mathbb{G}_t^2$ $+(1+u_i)\mathbb{G}_t^3 + (2+n_i)\mathbb{G}_1^2$	$\left((k+3)\mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_1^2\right) +$
	even: $\sum_{i=2,4}^{L} 1\mathbb{G}_1 + (n_i+2)\mathbb{G}_2 + (1+d_i)\mathbb{G}_t^2$ $+(1+u_i)\mathbb{G}_t^3 + (2+n_i)\mathbb{G}_2^2$	$\left(\mathbb{G}_1^{\lvert D\rvert}\right) + \left(\sum_{i=1}^{\lvert D\rvert}\left(\mathbb{G}_1^{u_i} + \mathbb{G}_1\right)\right)$
CredVerify	$(1+d_1)E + (3+u_1+d_L)E^2 + u_L E^3$ $+(4+n_1+d_L)\{\mathbb{G}_t\}+$	$\left(E^k + E^2 + 4E\right) +$
	$\sum_{i=1}^{L}((1+d_i)E^2 + (1+u_i)E^3 + (1+d_i)\{\mathbb{G}_t\})$	$\left(E + E^k + \mathbb{G}_2^{\lvert S\rvert} + \sum_{i=1}^{\lvert D\rvert}\left(\mathbb{G}_2^{\lvert S-d_i\rvert} + \mathbb{G}_2\right)\right)$

B.1 Computational Complexity

To analyze the efficiency of our DAC scheme, we consider the number of (multi)-exponentiations required for the IssueCred (issuing or delegating), CredProve (the showing of a credential), and CredVerify (verifying a credential). We summarize the following efficiency analysis comparing our DAC and the CDD scheme [CDD17] in Table 3. We use notations that are used in [CDD17], where d_i and u_i denote the amount of disclosed and undisclosed attributes at delegation level i, respectively, such that $n_i = d_i + u_i$; and $X\{\mathbb{G}_1^j\}, X\{\mathbb{G}_2^j\}$, and $X\{\mathbb{G}_t^j\}$ denote X j-multi-exponentiations in the respective group; $j = 1$ denotes a simple exponentiation. E^k denote a k-pairing product with $k = 1$ denoting a single pairing. Assume $z = |[k+1, k']|$ and $k'' = k'$ (the worst case), where k is length of message $\boldsymbol{M}$ and commitment $\boldsymbol{C}$ vectors and $n < t$ is size of a message (attributes) set M s.t. M_i includes n_i messages. Moreover, we assume that a n_i number of elements for each $j \in [z]$ can be delegated and put into $\mathsf{uk}_{k'}$. We summarize the efficiency analysis as follows, where we also count the cost of the $\boldsymbol{C}$ randomization in ChangeRep, but ignore the cost of proving knowledge of the secret key, as a Schnorr NIZK induces an insignificant cost:

CreateCred: CA creates the first level of the credential. To do this, CA runs Sign and a user runs ChangeRep/$\mathsf{uk}_{k'}$:

$$\left(\left(\sum_{i=1}^{k}(\mathbb{G}_1^{n_i} + \mathbb{G}_1)\right) + \mathbb{G}_1^2 + \mathbb{G}_1 + \mathbb{G}_1^{k+1} + \sum_{i=1}^{z} n_i\mathbb{G}_1 + \mathbb{G}_2\right) + \left((k+3)\mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_1^2 + \sum_{i=1}^{z} n_i\mathbb{G}_1\right)$$

IssueCred: Delegation of a credential includes running the ConvertSig, ChangeRel, and ChangeRep. We have:

$$\left(2\mathbb{G}_1^2\right) + \left(2(\mathbb{G}_1^n + \mathbb{G}_1) + \mathbb{G}_1^2\right) + \left((k+3)\mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_1^2\right).$$

If a further delegation is needed, then the randomization of $\mathsf{uk}_{k'}$ adds $\sum_{i=1}^{z} n_i\mathbb{G}_1$.

CredProve: Proving possession of a credential by a user includes ChangeRep, AggregateAcross, and OpenSubset: Let $D = (d_i)_{i\in[k]}$, we have:

$$\left((k+3)\mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_1^2\right) + \left(\mathbb{G}_1^{|D|}\right) + \left(\sum_{i=1}^{|D|}(\mathbb{G}_1^{u_i} + \mathbb{G}_1)\right)$$

CredVerify: The credential verification includes SPSEQ-UC.Verify (using CSCA.VerifyAcross): Let $S = \bigcup_i d_i$, where $i \in [k]$, we have

$$\left(E^k + E^2 + 4E\right) + \left(E + E^k + \mathbb{G}_2^{|S|} + \sum_{i=1}^{|D|}\left(\mathbb{G}_2^{|S-d_i|} + \mathbb{G}_2\right)\right)$$

Although we do not have a concept of delegation chain length, in order to compare our scheme with [CDD17], we assume that each commitment is the output of one delegation-level, e.g., if k = 2 such that $\boldsymbol{C} = (C_1, C_2)$, the $L = 2$ is delegation level. In other words, at each delegation level, we add one attribute set and the related commitment. Also, unlike [CDD17] where their underlying signature construction needs switching $\mathbb{G}_1$ and $\mathbb{G}_2$ of message space throughout, we do not need it. Note that *operations in* $\mathbb{G}_1$ *are faster than* $\mathbb{G}_2$ and also *the length of elements in* $\mathbb{G}_1$ *is smaller*. We only consider the number of (multi)exponentiations required to show a credential since this will be the most frequently executed operation. The result of CDD scheme is taken from Table 1 in [CDD17].

Credential size. The size only counts the cryptographic components of the credential; the metadata and attribute values are assumed to be the same for all systems. In particular, the credential size (σ, sk_p and pseudonym nym_p) in SPSEQ-UC is independent of the delegation chain length and number of attributes. In SPSEQ-UC, credentials have constant size which is four $\mathbb{G}_1$, one $\mathbb{G}_2$ and one $\mathbb{Z}_p$ element. Let $|\mathbb{G}_1| = |Z_p| = 256$ and $|\mathbb{G}_2| = 512$ in bit, we have a size of 1792 bits. While, in CDD the credential size grows linearly with the number of attributes and delegation levels. Also, the size of the related $\mathsf{uk}_{k'}$ in our scheme is $z \cdot 256$.

B.2 Communication Complexity

We analyze the communication complexity and the size of each element exchange involved in DAC. More precisely, the IssueCred protocol depends on the number of keys in $\mathsf{uk}_{k'}$ (if delegation is requested), while the CredProveprotocol is independent of the number of attributes, delegation levels and keys. In the CredProve, we have: $((k+5)|\mathbb{G}_1| + |\mathbb{G}_2| + Z_p)$, where k is the size of $\boldsymbol{C}$ (that is, the delegation level L) that we send for verification and Z_p with one $\mathbb{G}_1$ element that belong to the ZKPoK. In the IssueCred, we have $((k+3)|\mathbb{G}_1| + |\mathbb{G}_2| + k|Z_p| + z|\mathbb{G}_1|)$, where $z = |[k, k']|$ is the number of keys in this range. In CDD, this communication cost grows linearly with the number of attributes and delegation levels (see Table 4.) Here, for the CDD scheme, we take a proof generated from an *even Level-L* credential.

Table 4. Communication Complexity

Schemes	CredProve
Our DAC	$((k+5)\|\mathbb{G}_1\| + \|\mathbb{G}_2\| + Z_p)$
CDD [CDD17]	$(2L + \sum_{i=1,3}^{L-1}(n_i + u_i))\|\mathbb{G}_1\| +$ $(2L - 1 + \sum_{i=2,4}^{L}(n_i + u_i))\|\mathbb{G}_2\| + 2Z_p$

C Additional Definitions

C.1 Correctness of SPSEQ-UC

Subsequently, we state all the single correctness requirements.

Definition 15 (Correctness). *A* SPSEQ-UC *scheme for a set commitment scheme* SC *and a parameterized family of equivalence relations* $\mathbb{R}^\ell$ *for all* $\ell > 1$*, is correct if it satisfies the following conditions for all* t, λ*,* k, k' *with* $k \leq k' \leq \ell$*, for all* $\mathsf{pp} \in \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$*,* $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$*,* $\mathsf{pk}_u \in \mathsf{UKeyGen}(\mathsf{pp})$*, all* $\boldsymbol{M}$*,* $\boldsymbol{\rho}$*,* $\boldsymbol{T} \subseteq \boldsymbol{M}$*, all* $(\sigma, (\boldsymbol{C}, \boldsymbol{O}), \mathsf{uk}_{k'}) \in \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}, k', \mathsf{pk}_u; \boldsymbol{\rho})$*, any* $\boldsymbol{U}$ *with* $1 = \mathsf{SC.VerifySubset}(C_j, T_j, U_j)_{j \in k}$ *(for* U_j *being a subset opening) and* $1 = \mathsf{SC.Open}(C_i, T_i, U_i)_{j \in k}$ *(for* U_j *being an opening):*

Verification*: We have that:*

$$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \boldsymbol{C}, \sigma, (\boldsymbol{T}, \boldsymbol{U})) = \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1.$$

Change of set commitments representative: *For all* (μ, ψ)*,* $(\sigma', \mathsf{uk}'_{k'}, \chi) \in \mathsf{ChangeRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\boldsymbol{C}, \boldsymbol{O}), \sigma, \mu, \psi)$*, all* $(\boldsymbol{C}', \boldsymbol{O}') \leftarrow \mathsf{RndmzC}(\boldsymbol{C}, \boldsymbol{O}, \mu)$*,* $\mathsf{pk}'_u \leftarrow \mathsf{RndmzPK}(\mathsf{pk}, \psi, \chi)$ *and any* $\boldsymbol{U}'$ *s.t. either* $U'_j \leftarrow \mathsf{SC.OpenSubset}(C'_j, O'_j, T_j)$ *or* $U'_j = O'_j$ *we have:*

$$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}'_u, \boldsymbol{C}', \sigma', (\boldsymbol{T}, \boldsymbol{U}')) = 1 \text{ and } \boldsymbol{C}' \in [\boldsymbol{C}]_{\mathcal{R}^k}.$$

Signature conversion*: For all* $(\mathsf{pk}_{u'}, \mathsf{sk}_{u'}) \in \mathsf{UKeyGen}(\mathsf{pp})$*,* $\sigma' \leftarrow \mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma)$ *it holds that*

$$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_{u'}, \boldsymbol{C}, \sigma', (\boldsymbol{T}, \boldsymbol{U})) = 1.$$

Change of set commitments relation*: For any iterative application of* $\mathsf{pk}_{u_l} \in \mathsf{UKeyGen}(\mathsf{pp})$*,* $(\mathsf{uk}_{k''_l}, \sigma'_l) \leftarrow \mathsf{ChangeRel}(M_l, \sigma_{l-1}, \boldsymbol{C}, \mathsf{uk}_{k'_{l-1}}, k'')$ *for any* M_l*, with* $\boldsymbol{C}' = (\boldsymbol{C}, C_l \in \mathsf{SC.Commit}(M_l))$ *and* $\boldsymbol{M}' = (\boldsymbol{M}, M_l)$*, any* $\boldsymbol{U}'$ *s.t.* $U'_j \leftarrow \mathsf{SC.OpenSubset}(C'_j, O'_j, T_j)_{j \in l} \vee U'_j = O'_j$ *with* $l < k'' \leq k'$ *and* $\boldsymbol{T}' \subseteq \boldsymbol{M}'$*, we have:*

$$\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_{u_l}, \boldsymbol{C}', \sigma'_l, (\boldsymbol{T}', \boldsymbol{U}')) = 1$$

whenever $l \in [k+1, k']$ *and also we have* $[\boldsymbol{C}']_{\mathcal{R}^l}$*.*

C.2 Class-Hiding

By class-hiding, we mean that, for all k, $1 < k \leq \ell$, given two messages vectors $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ of size k, it should be hard to tell whether or not $\boldsymbol{C}_1 \in [\boldsymbol{C}_2]_{\mathcal{R}^k}$.

Definition 16 (Class-hiding). *A* SPSEQ-UC *scheme for parameterized equivalence relations* $\mathcal{R}^k$ *is class-hiding if for all* λ *and polynomial-length* $\ell(\lambda)$ *and all probabilistic polynomial-time (PPT) adversaries* $\mathcal{A}$*, there exists a negligible function* ϵ *such that:*

$$\Pr\begin{bmatrix} \mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, 1^\ell, 1^t); \boldsymbol{C}_1 \leftarrow (\mathbb{G}_1^*)^k; \\ \boldsymbol{C}_2^0 \leftarrow (\mathbb{G}_1^*)^k; \boldsymbol{C}_2^1 \leftarrow [\boldsymbol{C}_1]_{\mathcal{R}^k}; b \leftarrow \{0,1\}; \\ b' \leftarrow \mathcal{A}(\mathsf{pp}, \boldsymbol{C}_1, \boldsymbol{C}_2^b) : b' = b \end{bmatrix} \leq \frac{1}{2} + \epsilon(\lambda)$$

$\underline{\mathsf{ExpUnf}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda,\ell,t,u)}$:

- $u \leftarrow \mathcal{A}()$
- $Q := \emptyset; \mathcal{UL} := \emptyset, \mathsf{pp} \leftarrow \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$;
- $\mathcal{UL} \leftarrow \mathsf{UKeyGen}^u()$;
- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$;
- $((\mathsf{sk}_u^*, \mathsf{pk}_u^*), (\boldsymbol{C}^*, \boldsymbol{T}^*, \boldsymbol{U}^*), \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}}\left(\mathsf{vk}, \mathsf{pp}, \{\mathsf{pk}_u\}_{(\mathsf{pk}_u, \mathsf{sk}_u) \in \mathcal{UL}}\right)$

return:

$$\begin{pmatrix} \forall (\mathsf{pk}_u, \mathsf{sk}_u) \notin \mathcal{UL}, \forall (\boldsymbol{M}, k', \mathsf{pk}_u) \in Q : (\boldsymbol{M}, \boldsymbol{T}^*) \notin \mathcal{R}_{k'} \\ \wedge\ \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u^*, \boldsymbol{C}^*, \sigma^*, (\boldsymbol{T}^*, \boldsymbol{U}^*)) = 1 \end{pmatrix}$$

$\underline{\mathcal{O}^{\mathsf{Sign}}(\boldsymbol{M}, \boldsymbol{\rho}, k', \mathsf{pk}_u)}$:

- If $\ell \geq k' \geq k$:
- Then $(\sigma, \boldsymbol{C}, \mathsf{uk}_{k'}) \leftarrow \mathsf{Sign}(\mathsf{sk}, k', \boldsymbol{M}, \boldsymbol{\rho}, \mathsf{pk}_u)$
- $Q = Q \cup \{(\boldsymbol{M}, k', \mathsf{pk}_u)\}$,
- **return** $((\boldsymbol{C}, \boldsymbol{O}), \sigma, \mathsf{uk}_{k'})$
- Else **return** $\perp$

Fig. 6. Experiment $\mathsf{ExpUnfStat}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda, \ell, t)$.

D Proofs

D.1 Unforgeability of SPSEQ-UC

Our main technical result is to prove that our scheme satisfies unforgeability (Def. 3) in the generic group model (GGM) [Sho97] for asymmetric ("Type-3") bilinear groups (for which there are no efficiently computable homomorphisms between P and $\hat{P}$). In this model, the adversary is only given *handles* of group elements, which are just uniform random strings. To perform group operations, it uses an oracle to which it can submit handles and is given back the handle of the sum, inversion, etc of the group elements for which it submitted handles. Here, we will use the additional notation for the group law. Let analyze the unforgeability game in the GGM.

Theorem 4. *A generic adversary that computes at most q group operations and makes up to k queries to i ts signature oracle cannot win the semi-static game from Fig 1 for the scheme defined in 3.3 with probability greater than* $\frac{\left(o+q(4+\ell \cdot t)\right)^2 (2q+1)}{p}$.

Proof. First we consider an adversary against the static game defined in Fig 6 that only uses generic group operations on the group elements it receives. After getting the public parameter $(\alpha^o P, \alpha^o \hat{P})_{0 \leq o \leq t}$, a verification key $\left(X_0, \hat{X}_1 = x_1 \hat{P}, \ldots, \hat{X}_\ell = x_\ell \hat{P}\right)$, public keys $(P_1, \ldots, P_b)$, vector commitments $\left(C_1^{(i)}, \ldots, C_{k^{(i)}}^{(i)}\right)_{i=1}^q$ and signatures $\left(Z_i, S_i, \hat{S}_i, T_i\right)_{i=1}^q$ computed with randomness s_i on queries

$$\left(\left(M_1^{(i)}, \ldots, M_{k^{(i)}}^{(i)}\right), \left(\rho_1^{(i)}, \ldots, \rho_{k^{(i)}}^{(i)}\right), k^{(i)\prime}, \mathsf{pk}^{(i)}\right)_{i=1}^q$$, with opening: table keys $\left(\left(U_{j,o}^{(i)}\right)\right)_{j \in [k^{(i)}+1, k^{\prime(i)}], o \in [t]}$

the adversary outputs a public user key $\mathsf{pk}^{(q+1)}$, a vector of commitments $\left(C_1^{(*)}, \ldots, C_{k^{(*)}}^{(*)}\right)$ and a corresponding signature $\left(Z^*, S^*, \hat{S}^*, T^*\right)$. As it must compute any new group element by combining received group elements, it must choose coefficients that we will represent here by using greek letters, which define

$$C_h^* = \kappa^{(h)} P + \sum_{j=1}^{q} \left(\kappa_{z,j}^{(h)} Z_j + \kappa_{s,j}^{(h)} S_j + \kappa_{t,j}^{(h)} T_j + \sum_{o=0}^{t} \sum_{m=k^{(j)}+1}^{k^{\prime(j)}} \kappa_{m,j,o}^{(h)} U_{m,o}^{(j)}\right) + \sum_{o=1}^{t} \kappa_{a,o}^{(h,q+1)} a^o P + \kappa_{x,0}^{(h)} X_0 + \sum_{u=1}^{b} \kappa_{\mathsf{pk},u}^{(h)} P_a$$

$$Z^* = \zeta P + \sum_{j=1}^{q} \Big(\zeta_{z,j} Z_j + \zeta_{s,j} S_j + \zeta_{t,j} T'_j + \sum_{o=0}^{t} \sum_{m=k'^{(j)}+1}^{k'^{(j)}} \zeta_{m,j,o} U_{m,o}^{(j)} \Big) + \sum_{o=1}^{t} \zeta_{a,o} a^o P + \zeta_{x,0} X_0 + \sum_{u=1}^{b} \zeta_{\mathsf{pk},u} P_a$$

$$S^* = \sigma P + \sum_{j=1}^{q} \Big(\sigma_{z,j} Z_j + \sigma_{s,j} S_j + \sigma_{t,j} T_j + \sum_{o=0}^{t} \sum_{m=k'^{(j)}+1}^{k'^{(j)}} \sigma_{m,j,o} U_{m,o}^{(j)} \Big) + \sum_{o=1}^{t} \sigma_{a,o} a^o P + \sigma_{x,0} X_0 + \sum_{u=1}^{b} \sigma_{\mathsf{pk},u} P_a \Big)$$

$$T^* = \tau P + \sum_{j=1}^{q} \Big(\tau_{z,j} Z_j + \tau_{s,j} S_j + \tau_{t,j} T_j + \sum_{o=1}^{t} \sum_{m=k'^{(j)}+1}^{k'^{(j)}} \tau_{m,j,o} U_{m,o}^{(j)} \Big) + \sum_{o=1}^{t} \tau_{a,o} a^o P + \sum_{u=1}^{b} \tau_{\mathsf{pk},u} P_a \Big)$$

$$\mathsf{pk}^{(i)} = \psi^{(i)} P + \sum_{j=1}^{i-1} \Big(\psi_{z,j}^{(i)} Z_j + \psi_{s,j}^{(i)} S_j + \psi_{t,j}^{(i)} T_j + \sum_{o=0}^{t} \sum_{m=k'^{(j)}+1}^{k'^{(j)}} \psi_{m,j,o}^{(i)} U_{m,o}^{(j)} \Big) + \sum_{o=1}^{t} \psi_{a,o}^{(i)} a^o P + \psi_{x,0}^{(i)} X_0 + \sum_{u=1}^{b} \psi_{\mathsf{pk},u}^{(i)} P_a$$

$$\hat{S}^* = \phi \hat{P} + \sum_{j=0}^{\ell} \phi_{x,j} \hat{X}_j + \sum_{j=1}^{q} \phi_{s,j} \hat{S}_j + \sum_{o=1}^{t} \phi_{a,o} a^o \hat{P}$$

Using this, we can write, for all $1 \leq i \leq q$, the discrete logarithms $c_j^{(i)}, z_i$ and t_i in basis P of the elements $C_j^{(i)} = \prod_{e \in M_j^{(i)}} (\alpha - e) P$, $Z_i = \dfrac{\sum_{j=1}^{k^{(i)}} x_j C_j^{(i)}}{s_i}$, $T_i = X_1 s_i + x_0 \mathsf{pk}^{(i)}$ and $U_{m,o}^{(i)} = \frac{a^o x_m}{s_i} P$ from the oracle answers.

$$c_j^{(i)} = \rho_j^{(i)} \prod_{e \in M_j^{(i)}} (\alpha - e) P \tag{1}$$

$$z_i = \frac{1}{s_i} \left(\sum_{h=1}^{k^{(i)}} x_h c_h^{(i)} \right) \tag{2}$$

$$t_i = x_1 s_i + x_0 \mathsf{pk}^{(i)} \tag{3}$$

$$u_{m,o}^{(i)} = \frac{a^o x_m}{s_i} \tag{4}$$

A successful forgery $(Z^*, S^*, \hat{S}^*, T^*)$ on $\left(\mathsf{pk}^{(q+1)}, (C_1^*, \ldots, C_{k^{(q+1)}}^*)\right)$ satisfies the verification equations

$$
\begin{aligned}
e(Z^*, \hat{S}^*) &= \prod_{h=1}^{k^{(q+1)}} e(C_h^*, \hat{X}_h) \\
e(P, \hat{S}^*) &= e(S^*, \hat{P}) \\
e(T^*, \hat{P}) &= e(S^*, \hat{X}_1) e(\mathsf{pk}^{(q+1)}, \hat{X}_0)
\end{aligned}
$$

We interpret these values as multivariate rational fractions in variables $x_0, x_1, \ldots, x_\ell, s_1, \ldots, s_q, a, p_1, \ldots, p_b$.

Using the coefficients defined above and considering the logarithms in base $e(P, \hat{P})$ we obtain:

$$
\begin{aligned}
&\Big(\zeta + \sum_{j=1}^{q} \big(\zeta_{z,j} z_j + \zeta_{s,j} s_j + \zeta_{t,j} t_j + \sum_{o=0}^{t} \sum_{m=k^{(j)}+1}^{k'^{(j)}} \zeta_{m,j,o} u_{m,o}^{(j)}\big) + \\
&\sum_{o=1}^{t} \zeta_{a,o} a^o + \zeta_{x,0} x_0 + \sum_{u=1}^{b} \zeta_{\mathsf{pk},u} p_a\Big) \\
&\Big(\phi + \sum_{j=0}^{\ell} \phi_{x,j} x_j + \sum_{j=1}^{q} \phi_{s,j} s_j + \sum_{o=1}^{t} \phi_{a,o} a^o\Big) = \sum_{h=1}^{k^{(q+1)}} x_h c_h^*
\end{aligned} \tag{5}
$$

$$
\begin{aligned}
&\sigma + \sum_{j=1}^{q} \big(\sigma_{z,j} z_j + \sigma_{s,j} s_j + \sigma_{t,j} t_j + \sum_{o=0}^{t} \sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}} \sigma_{m,j,o} u_{m,o}^{(q+1)}\big) \\
&+ \sum_{o=1}^{t} \sigma_{a,o} a^o + \sigma_{x,0} x_0 + \sum_{u=1}^{b} \sigma_{\mathsf{pk},u} p_a \\
=&\phi + \sum_{j=0}^{\ell} \phi_{x,j} x_j + \sum_{j=1}^{q} \phi_{s,j} s_j + \sum_{o=1}^{t} \phi_{a,o} a^o
\end{aligned} \tag{6}
$$

$$
\begin{aligned}
&\tau + \sum_{j=1}^{q} \big(\tau_{z,j} z_j + \tau_{s,j} s_j + \tau_{t,j} t_j + \sum_{o=0}^{t} \sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}} \tau_{m,j,o} u_{m,o}^{(j)}\big) \\
&+ \sum_{o=1}^{t} \tau_{a,o} a^o + \tau_{x,0} x_0 + \sum_{u=1}^{b} \tau_{\mathsf{pk},u} p_a \\
=&x_1 \Big(\sigma + \sum_{j=1}^{q} \big(\sigma_{z,j} z_j + \sigma_{s,j} s_j + \sigma_{t,j} t_j + \sum_{o=0}^{t} \sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}} \sigma_{m,j,o} u_{m,o}^{(q+1)}\big) \\
&+ \sum_{o=1}^{t} \sigma_{a,o} a^o + \sigma_{x,0} x_0 + \sum_{u=1}^{b} \sigma_{\mathsf{pk},u} p_a\Big) \\
&+ x_0 \Big(\psi^{(q+1)} + \sum_{j=1}^{q} \big(\psi_{z,j}^{(q+1)} z_j + \psi_{s,j}^{(q+1)} s_j + \psi_{t,j}^{(q+1)} t_j \\
&+ \sum_{o=0}^{t} \sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}} \psi_{m,j,o}^{(q+1)} u_{m,o}^{(j)}\big) \\
&+ \sum_{o=1}^{t} \psi_{a,o}^{(q+1)} a^o + \psi_{x,0}^{(q+1)} x_0 + \sum_{u=1}^{b} \psi_{\mathsf{pk},u}^{(q+1)} p_a\Big)
\end{aligned} \tag{7}
$$

We follow the standard proof technique for results in the generic group model and now consider an "ideal" game in which the challenger treats all the (handles of) group elements as elements of

$\mathbb{Z}_p(s_1, \ldots, s_q, x_0, x_1, \ldots, x_\ell, a, p_1, \ldots, p_b)$, that is, rational fractions whose indeterminates represent the secret values chosen by the challenger.

We first show that in the ideal game if the adversary's output satisfies the verification equations, then the first winning condition is not satisfied, which demonstrates that the ideal game cannot be won. We then compute the statistical distance from the adversary's point of view between the real and the ideal game at the end of the proof.

In the ideal game we thus interpret the three equalities (5), (6) and (7) as polynomial equalities over the field $\mathbb{Z}_p(s_1, \ldots, s_q, x_0, x_1, \ldots, x_\ell, a, p_1, \ldots, p_b)$. More precisely, we consider the equalities in the ring $\mathbb{Z}_p(s_1, \ldots, s_q)[x_0, x_1, \ldots, x_\ell, a, p_1, \ldots, p_b]$, that is, the polynomial ring with $x_1, \ldots, x_\ell, a, p_1, \ldots, p_b$ as indeterminates over the field $\mathbb{Z}_p(s_1, \ldots, s_q)$. (Note that this interpretation is possible because neither any x_i's and p_u nor a never appear in the denominators of any expressions.) As one of our proof techniques, we will also consider the equalities over the ring factored by $(x_0, x_1, \ldots, x_\ell)$, the ideal generated by the x_i's:[13]

$$\begin{aligned} &Z_p(s_1, \ldots, s_q)[x_0, x_1, \ldots, x_\ell, a, p_1, \ldots, p_n]/(x_0, x_1, \ldots, x_\ell) \\ \cong &\mathbb{Z}_p(s_1, \ldots, s_q)[a, p_1, \ldots, p_n] \ . \end{aligned}$$

From (2) and (3), over this quotient we have and $t_i = z_i = u^{(j)}_{m,o} = 0$ and thus (5)–(7) become

$$\begin{aligned} &\left(\zeta + \sum_{j=1}^{q} \zeta_{s,j} s_j + \sum_{o=1}^{t} \zeta_{a,o} a^o + \sum_{u=1}^{b} \zeta_{\mathsf{pk},u} p_a\right) \\ &\left(\phi + \sum_{j=1}^{q} \phi_{s,j} s_j + \sum_{o=1}^{t} \phi_{a,o} a^o\right) = 0 \qquad (8) \\ &\sigma + \sum_{j=1}^{q} \sigma_{s,j} s_j + \sum_{o=1}^{t} \sigma_{a,o} a^o + \sum_{u=1}^{b} \sigma_{\mathsf{pk},u} p_a \\ &= \phi + \sum_{j=1}^{q} \phi_{s,j} s_j + \sum_{o=1}^{t} \phi_{a,o} a^o \qquad (9) \\ &\tau + \sum_{j=1}^{q} \tau_{s,j} s_j + \sum_{o=1}^{t} \tau_{a,o} a^o + \sum_{u=1}^{b} \tau_{\mathsf{pk},u} p_a = 0 \qquad (10) \end{aligned}$$

By looking the coefficients of the monomials s_j's, p_u's, a^o's and 1 of (10), we deduce:

$$\forall j, o, u : \tau = \tau_{s,j} = \tau_{a,o} = \tau_{\mathsf{pk},u} = 0 \qquad (11)$$

And by looking the coefficients of the $s_j, \frac{1}{s_j}, a^o$'s of (9), we deduce:

$$\sigma = \phi, \forall o : \sigma_{a,o} = \phi_{a,o}, \qquad \forall j : \sigma_{s,j} = \phi_{s,j} \qquad \forall u : \sigma_{\mathsf{pk},u} = 0 \qquad (12)$$

[13] Considering an equation of rational fractions over this quotient can also be seen as simply setting $\forall i, x_i = 0$. Everything we infer about the coefficients from these modified equations is also valid for the original equation, since these must hold for all values $(s_1, \ldots, s_q, x_0, x_1, \ldots x_\ell, a, p_1, \ldots, p_b)$ and so in particular for $(s_1, \ldots, s_q, 0, 0, \ldots, 0, a, p_1, \ldots, p_b)$.

Yet another interpretation when equating coefficients in equations modulo $(x_0, x_1, \ldots, x_\ell)$ is that one equates coefficients only of monomials that do not contain any x_i.

Now we can reuse (12) in (6) and look the equation modulo (x_0).

$$\sum_{j=1}^{q}\big(\sigma_{z,j}z_j+\sigma_{t,j}s_jx_1+ \sum_{o=0}^{t}\sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}}\sigma_{m,j,o}u_{m,o}^{(q+1)}\big) \mod (x_0)=\sum_{j=1}^{\ell}\phi_{x,j}x_j \tag{13}$$

By looking the coefficient in the monomials x_j's, for $j>0$ and because for all j, $deg_{s_j}(z_j)=deg_{s_j}(u_{m,j,o})=-1$, we deduce:

$$\forall j>0:\phi_{x,j}=0. \tag{14}$$

Then the equation (13) becomes:

$$\sum_{j=1}^{q}\big(\sigma_{z,j}\left(\sum_{m=2}^{k^{(j)}}\frac{x_m\prod_{e\in M_m^{(j)}}(a-e)}{s_j}\right)+\sigma_{t,j}s_jx_1+ \sum_{o=0}^{t}\sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}}\sigma_{m,j,o}u_{m,o}^{(q+1)}\big)=0 \tag{15}$$

By looking all the monomials in x_1s_j, we deduce:

$$\forall j:\sigma_{t,j}=0 \tag{16}$$

Now, for all j, we look all the monomials of degree -1, in s_j, we deduce:

$$\forall j:\sigma_{z,j}\left(\sum_{m=1}^{k^{(j)}}\frac{x_m\prod_{e\in M_m^{(j)}}(a-e)}{s_j}\right)+\sum_{o=0}^{t}\sum_{m=k^{(j)}+1}^{k'^{(j)}}\sigma_{m,j,o}\frac{a^ox_m}{s_j}=0 \tag{17}$$

Then, by looking the monomials in a^ox_j for all $j>k^{(j)}$:

$$\forall m,j,o:\sigma_{m,j,o}=0 \tag{18}$$

Then (17) becomes:

$$\forall j:\sigma_{z,j}\left(\sum_{m=1}^{k^{(j)}}\frac{x_m\prod_{e\in M_m^{(j)}}(a-e)}{s_j}\right)=0 \tag{19}$$

Then without any loss of generalities, we deduce:

$$\forall j:\sigma_{z,j}=0 \tag{20}$$

Now we look the equation (5) modulo $(x_1,\ldots,x_\ell)$:

$$\left(\phi+\phi_{x,0}x_0+\sum_{j=1}^{q}\phi_{s,j}s_j+\sum_{o=1}^{t}\phi_{a,o}a^o\right)\Big(\zeta+\sum_{j=1}^{q}\big(\zeta_{s,j}s_j+\zeta_{t,j}t_j\big)+ \sum_{o=1}^{t}\zeta_{a,o}a^o+\zeta_{x,0}x_0+\sum_{u=1}^{b}\zeta_{\mathsf{pk},u}p_a\Big) \mod (x_1,\ldots,x_\ell)=0$$

Now, because $\hat{S}^* \neq 0$, we can deduce: $\phi + \phi_{x,0}x_0 + \sum_{j=1}^{q} \phi_{s,j}s_j + \sum_{o=1}^{t} \phi_{a,o}a^o \neq 0$, then because

$$t_j \mod (x_1, \ldots, x_\ell) = \frac{x_0}{s_j}\mathsf{pk}^{(j)} \mod (x_1, \ldots, x_\ell)$$

we have:

$$\Big(\zeta + \sum_{j=1}^{q} \left(\zeta_{s,j}s_j + \zeta_{t,j}\tfrac{x_0}{s_j}\mathsf{pk}^{(j)}\right) + \sum_{o=1}^{t} \zeta_{a,o}a^o + \zeta_{x,0}x_0 + \sum_{u=1}^{b} \zeta_{\mathsf{pk},u}p_a\Big) \mod (x_1, \ldots, x_\ell) = 0 \tag{21}$$

Then, by looking constant coefficient in x_0, we deduce:

$$\forall j : \zeta_{s,j} = 0, \qquad \forall o : \zeta_{a,o} = 0, \qquad \zeta = 0 \qquad \forall u : \zeta_{\mathsf{pk},u} = 0 \tag{22}$$

Then we by noticing for all j, $\mathsf{pk}^{(j)}$ is constant in s_j. By looking coefficient constant in s_j, but of degree 1 in x_0.

$$\zeta_{x,0} = 0. \tag{23}$$

Now, let's look equation (7) modulo (x_0, x_1)

$$\sum_{j=1}^{q} \left(\tau_{z,j}z_j + \sum_{o=0}^{t} \sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}} \tau_{m,j,o}u_{m,o}^{(j)}\right) \mod (x_0, x_1) = 0$$

Now, for all j, we look all the monomials of degree -1, in s_j, and degree 0 in s_k for $k > j$:

$$\forall j : \tau_{z,j}\left(\sum_{m=2}^{k^{(j)}} \frac{x_m \prod_{e \in M_m^{(j)}}(a-e)}{s_j}\right) + \sum_{o=0}^{t} \sum_{m=k^{(j)}+1}^{k'^{(j)}} \tau_{m,j,o}\frac{a^o x_m}{s_j} \mod (x_0, x_1) = 0 \tag{24}$$

Then, by looking the monomials in $\frac{a^o x_m}{s_j}$ for $m > k^{(j)}$:

$$\forall m, j, o : \tau_{m,j,o} = 0 \tag{25}$$

Then (7) modulo (x_0) becomes:

$$\begin{aligned}&\sum_{j=1}^{q} \left(\tau_{z,j}z_j + \tau_{t,j}x_1 s_j\right) \mod (x_0)\\ =&x_1\left(\sigma + \sum_{j=1}^{q} \left(\sigma_{s,j}s_j\right) + \sum_{o=1}^{t} \sigma_{a,o}a^o\right)\end{aligned} \tag{26}$$

By looking the monomials constant in s_i, $\forall i$, we deduce:

$$\forall o : \sigma = \sigma_{a,o} = 0 \tag{27}$$

(26) becomes thus:

$$\sum_{j=1}^{q}\left(\sigma_{z,j}z_j+\sum_{o=0}^{t}\sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}}\sigma_{m,j,o}u_{m,o}^{(q+1)}\right)+\sigma_{x,0}x_0=\phi_{x,0}x_0 \tag{28}$$

By looking monomials constant in s_i, $\forall i$:

$$\sigma_{x,0}=\phi_{x,0} \tag{29}$$

Then, by using (27),in (26), we deduce:

$$\sum_{j=1}^{q}\left(\tau_{z,j}z_j+\tau_{t,j}x_1s_j\right)\mod(x_0)=x_1\left(\sum_{j=1}^{q}\sigma_{s,j}s_j\right) \tag{30}$$

If we look in this equation for all j, all the monomials of degree -1, in s_j, and degree 0 in s_k for $k>j$. We deduce

$$\forall j:\tau_{z,j}z_j=0$$

Then without any loss of generality, we can assume:

$$\forall j:\tau_{z,j}=0 \tag{31}$$

Then by looking monomials in x_1s_j, for all j, we deduce:

$$\forall j:\sigma_{s,j}=\tau_{t,j} \tag{32}$$

Then, (7) becomes:

$$\sum_{j=1}^{q}\sigma_{s,j}t_j+\tau_{x,0}x_0=x_1\left(\sum_{j=1}^{q}\sigma_{s,j}s_j+\sigma_{x,0}x_0\right)+x_0\mathsf{pk}^{(q+1)} \tag{33}$$

Now, if we look the monomial x_0x_1, we deduce

$$\sigma_{x,0}=0 \tag{34}$$

And (29) implies:

$$\phi_{x,0}=0 \tag{35}$$

Now, let's look (5) by using (35), (14), (12):

$$\left(\sum_{j=1}^{q}\left(\zeta_{z,j}z_j+\zeta_{t,j}t_j+\sum_{o=0}^{t}\sum_{m=k^{(j)}+1}^{k'^{(j)}}\zeta_{m,j,o}u_{m,o}^{(j)}\right)\right)\left(\sum_{j=1}^{q}\phi_{s,j}s_j\right)=\sum_{h=1}^{k^{(q+1)}}x_hc_h^* \tag{36}$$

Let i_0 be the maximum of the i's such that $\phi_i\neq 0$.
Then $\left(\sum_{j=1}^{q}\phi_{s,j}s_j\right)=\left(\sum_{j=1}^{i_0}\phi_{s,j}s_j\right)$ is of degree 1 in s_{i_0} and in s_{i_1}.

Now we can notice that in $\sum_{h=1}^{k^{(q+1)}} x_h c_h^*$, there is neither monomial of degree -1 in s_i and of degree 1 in s_k with $k \neq i$ nor monomials in $s_i s_k$, nor in s_i^2

Because, $\Big(\sum_{j=1}^{i_0} \phi_{s,j} s_j \Big)$ is of degree 1 in s_{i_0}.

We deduce that the left term has no term of degree 1 or -1 in any indeterminate s_i in $\{s_1, \ldots, s_q\}$.

In particular, there is no monomials in $x_1 s_i$ in this term,

Then

$$\forall i : \zeta_{t,i} t_i = 0 \tag{37}$$

$$\forall j \neq i_0 : \zeta_{z,j} z_j + \sum_{o=0}^{t} \sum_{m=k^{(j)}+1}^{k'^{(j)}} \zeta_{m,j,o} u_{m,o}^{(j)} = 0 \tag{38}$$

Then (36) becomes:

$$\big(\zeta_{z,i_0} z_{i_0} + \sum_{o=0}^{t} \sum_{m=k^{(j)}+1}^{k'^{(i_0)}} \zeta_{m,j,o} u_{m,o}^{(i_0)}\big)\big(\sum_{j=1}^{i_0} \phi_{s,j} s_j\big) = \sum_{h=1}^{k^{(q+1)}} x_h c_h^* \tag{39}$$

By noticing $\sum_{h=1}^{k^{(q+1)}} x_h c_h^*$ has no monomial in $s \frac{s_j}{s_{i_0}}$, for $j < i_0$, we deduce:

$$\forall j < i_0 : \phi_{s,j} = 0 \tag{40}$$

We can now transform (39) in:

$$\zeta_{z,i_0} \sum_{m=1}^{k^{(i_0)}} x_m c_m^{(i_0)} + \sum_{o=0}^{t} \sum_{m=k^{(j)}+1}^{k'^{(i_0)}} \zeta_{m,j,o} x_m a^o = \sum_{m=1}^{k^{(q+1)}} x_m c_m^* \tag{41}$$

We can deduce by looking for all the monomials in x_i:

$$\forall m \leq k^{(i_0)} : \zeta_{z,i_0} c_m^{(i_0)} = c_m^* \tag{42}$$

$$\forall m \in \{k^{(i_0)} + 1, \ldots, k'^{(i_0)}\} : \sum_{o=0}^{t} \zeta_{m,j,o} a^o = c_m^* \tag{43}$$

Now, we can use all the equalities found to deduce from (7):

$$\begin{aligned}
&\sigma_{s,i_0} t_{i_0} + \tau_{x,0} x_0 \\
=&x_1 \sigma_{s,i_0} s_{i_0} \\
&+ x_0 \Bigg(\psi^{(q+1)} + \sum_{j=1}^{q} \big(\psi_{z,j}^{(q+1)} z_j + \psi_{s,j}^{(q+1)} s_j + \psi_{t,j}^{(q+1)} t_j \\
&+ \sum_{o=0}^{t} \sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}} \psi_{m,j,o}^{(q+1)} u_{m,o}^{(j)} \big) \\
&+ \sum_{o=1}^{t} \psi_{a,o}^{(q+1)} a^o + \psi_{x,0}^{(q+1)} x_0 + \sum_{u=1}^{b} \psi_{\mathsf{pk},u}^{(q+1)} p_a \Bigg)
\end{aligned} \tag{44}$$

It becomes:

$$\begin{aligned}
&\sigma_{s,i_0} x_0 \mathsf{pk}^{(i_0)} + \tau_{x,0} x_0 = \\
&x_0 \Bigg(\psi^{(q+1)} + \sum_{j=1}^{q} \big(\psi_{z,j}^{(q+1)} z_j + \psi_{s,j}^{(q+1)} s_j + \psi_{t,j}^{(q+1)} t_j \\
&+ \sum_{o=0}^{t} \sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}} \psi_{m,j,o}^{(q+1)} u_{m,o}^{(j)} \big) \\
&+ \sum_{o=1}^{t} \psi_{a,o}^{(q+1)} a^o + \psi_{x,0}^{(q+1)} x_0 + \sum_{u=1}^{b} \psi_{\mathsf{pk},u}^{(q+1)} p_a \Bigg)
\end{aligned} \tag{45}$$

Then

$$\sigma_{s,i_0} \mathsf{pk}^{(i_0)} + \tau_{x,0} = \mathsf{pk}^{(q+1)}. \tag{46}$$

Because, the adversary should output the secret key associated to $\mathsf{pk}^{(q+1)}$. $\mathsf{pk}^{(q+1)} = \psi P$. Then, recall that because $\sigma_{s,i_0} \neq 0$

$$\mathsf{pk}^{(i_0)} = \frac{(\psi - \tau_{x,0})}{\sigma_{s,i_0}}. \tag{47}$$

We deduce that $\mathsf{pk}^{(i_0)} \notin \mathcal{UL}$.

It implies that $(\boldsymbol{M_{i_0}}, \boldsymbol{T}^*) \notin \mathcal{R}_{k^{(i_0)}}$, thus $\exists j_0 \in \{1, \ldots, k^{(i_0)}\}$, such that $T_{j_0} \not\subset M_{j_0}^{(i_0)}$. Then, it exists $e' \in T_{j_0} \setminus M_{j_0}^{(i_0)}$ Now, let's look how W_h^* has been built by the adversary:

$$\begin{aligned}
W_{j_0}^* = \omega P + \sum_{j=1}^{q} \big(&\omega_{z,j} Z_j + \omega_{s,j} S_j + \omega_{t,j} T_j + \sum_{o=0}^{t} \sum_{m=k^{(j)}+1}^{k'^{(j)}} \\
&\omega_{m,j,o} U_{m,o}^{(j)} \big) + \sum_{o=1}^{t} \omega_{a,o}^{(h,q+1)} a^o P + \omega_{x,0}^{(h)} X_0 + \sum_{u=1}^{b} \omega_{\mathsf{pk},u}^{(h)} P_a
\end{aligned}$$

Because VerifySubset outputs 1 then:

$$\begin{aligned}
\Bigg(\omega + \sum_{j=1}^{q} \big(&\omega_{z,j} z_j + \omega_{s,j} s_j + \omega_{t,j} t_j + \sum_{o=0}^{t} \sum_{m=k^{(j)}+1}^{k'^{(j)}} \omega_{m,j,o} u_{m,o}^{(j)} \big) \\
&+ \sum_{o=1}^{t} \omega_{a,o}^{(h,q+1)} a^o + \omega_{x,0}^{(h)} x_0 + \sum_{u=1}^{b} \omega_{\mathsf{pk},u}^{(h)} p_a \Bigg) \\
&\Bigg(\prod_{e \in T_{j_0}^*} (\alpha - e) \Bigg) = \zeta_{z,i_0} \prod_{e \in M_{j_0}^{(i_0)}} (\alpha - e)
\end{aligned} \tag{48}$$

This equation modulo $(a - e')$:

$$\zeta_{z,i_0} \prod_{e \in M_j^{(i_0)}} (a - e) \mod (a - e') = 0$$

Then, we have a contradiction, because $e' \notin M_j^{(i_0)}$.

We have thus shown that in the "ideal" model, the attacker cannot win the game. It remains to upper-bound the statistical distance from the adversary point of view between these two models.

Difference between ideal and real game. We can upper bound the degree of the denominators of all the rational fractions by $(2q + t + 1)$. If the adversary computes at most o group operations, then there are at most $o + q(4 + \ell \cdot t)$ group elements. Using the union bound, we conclude that the adversary can distinguish the two models with probability at most $\frac{\left(o+q(4+\ell\cdot t)\right)^2(2q+t+1)}{p}$.

Difference between static and adaptative game. If we consider a deterministic adaptative adversary (which can make adaptative corruption like in the real game).

If there is no collision before the first corruption (it happens with probability $\frac{\left(o+q(4+\ell\cdot t)\right)^2(2q+t+1)}{p}$), the challenger can guess the traitor and then can make it static.

It can do it for c corruption if there is no collision, and this could happen with probability $\frac{c\left(o+q(4+\ell\cdot t)\right)^2(2q+t+1)}{p.}$

D.2 Privacy Notions of SPSEQ-UC

We now prove that our SPSEQ-UC construction from Section 3.3 is Origin-hiding (Def. 4) and provides Conversion-privacy (Def. 5) and Derivation-privacy (Def. 6).

Lemma D1 (Origin-hiding) *The construction described in Section 3.3 is Origin-hiding.*

Proof. Origin-hiding of ChangeRep follows from the perfect adaptation of SPSEQ [FHS19]. The only main difference here is the additional element T in a signature as well as elements $(\mathsf{pk}_u, \mathsf{uk}_{k'})$ which we show that they are correctly randomized in both algorithms. Let $\boldsymbol{C} \in (\mathbb{G}_1^*)^k$, $\boldsymbol{T} \subseteq \boldsymbol{M}$, any $\boldsymbol{O}, \boldsymbol{U}$ s.t. $\mathsf{SC.Open}(C_j, M_j, O_j)_{j\in k} = 1$, $\mathsf{pk}_u \in \mathbb{G}_1$, and $(x_0, x_1, \ldots, x_\ell) \leftarrow (Z_p^*)^\ell$ be such that $\mathsf{vk} = ((\hat{P}^{x_i})_{i\in[0,\ell]}, P^{x_0})$. For some $y \in Z_p^*$, a signature $(Z, Y, \hat{Y}, T) \in \mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ satisfying $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \boldsymbol{C}, (Z, Y, \hat{Y}, T), (\boldsymbol{T}, \boldsymbol{U})) = 1$ along with $\mathsf{uk}_{k'}$ is of the form

$$\sigma = \left(\left(\prod_{i=1}^{k} C_i^{x_i} \right)^{y^{-1}}, P^y, \hat{P}^y, P^{x_1 \cdot y} \cdot \mathsf{pk}_u^{x_0} \right).$$

For $\mu, \psi \in Z_p^*$, $\mathsf{ChangeRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\boldsymbol{C}, \boldsymbol{O}), (Z, Y, \hat{Y}, T), \mu, \psi)$ outputs

$$\sigma' = \left(\left(\prod_{i=1}^{k} C_i^{\mu \cdot x_i} \right)^{y^{-1}\cdot\psi}, P^{y\cdot\psi}, \hat{P}^{y\cdot\psi}, P^{x_1\cdot y\cdot\psi} \cdot X_0^{\psi(\mathsf{sk}_u+\chi)} \right),$$

which is a uniformly random element σ' in $\mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$. That is, all elements of the signature σ' are perfectly randomized using randomness μ, ψ, χ and conditioned on $\mathsf{Verify}(\mathsf{vk}, (\mathsf{pk}_u \cdot P^\chi)^\psi, \boldsymbol{C}^\mu, \sigma', (\boldsymbol{T}, \boldsymbol{U})) = 1$. Now we show that $\mathsf{uk}_{k'}$ (in the case that it is requested for further delegation), and pk_u are also randomized perfectly using ψ, χ and μ. For all k', an update key $\mathsf{uk}_{k'} \in (\mathbb{G}_1^*)^{[k+1,k']}$ s.t. $\mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$ and $\mathsf{pk}_u \in \mathbb{G}_1^*$ are the from

$$\mathsf{uk}_{k'} = \left(\mathsf{usign}_j = \left((P^{\alpha^i})^{x_j} \right)^{y^{-1}}_{j\in[k+1,k']\wedge i\in[t]} \right) \text{ and } \mathsf{pk}_u = P^{\mathsf{sk}_u}.$$

For $\mu, \psi \in Z_p^*$, ChangeRep outputs the following form, so we have

$$\mathsf{uk}'_{k'} = \left((\mathsf{usign}_j)^{\psi^{-1}\cdot\mu}\right)_{j\in[k+1,k']} \text{ and } \mathsf{pk}'_u = P^{(\mathsf{sk}_u+\chi)\cdot\psi},$$

where $\chi \leftarrow Z_p^*$ is randomness selected locally for RndmzPK. It is not difficult to see that all elements of the $\mathsf{uk}_{k'}$ are distributed as expected and also pk_u is perfectly randomized with ψ, χ and represents a uniform element in $\mathbb{G}_1^*$. So, ChangeRep clearly produces signatures with the same distribution as Sign:

$$(\sigma, \boldsymbol{C}, \mathsf{uk}_{k'}, \mathsf{pk}_u) \approx (\sigma', \boldsymbol{C}', \mathsf{uk}'_{k'}, \mathsf{pk}'_u)$$

Lemma D2 (Conversion-privacy) *The construction described in Section 3.3 provides Conversion-privacy.*

First of all, let us assume that $\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma)$ includes $[\mathsf{SendConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \sigma) \leftrightarrow \mathsf{ReceiveConvertSig}(\mathsf{vk}, \mathsf{sk}_{u'})] \rightarrow \sigma'$, where σ' is a valid signature.

Proof. For all $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$, $(\mathsf{sk}_u, \mathsf{pk}_u) \in \mathsf{UKeyGen}$, $\boldsymbol{C}, \boldsymbol{T}, \boldsymbol{M}, \boldsymbol{U}, \boldsymbol{O}$ s.t. $\mathsf{SC.Open}(C_j, M_j, O_j)_{j\in k} = 1$ and σ. If $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \boldsymbol{C}, \sigma, (\boldsymbol{T}, \boldsymbol{U})) = 1$. The signature σ in $\mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ is the form of:

$$\sigma = \left(\left(\prod_{i=1}^{k} C_i^{x_i}\right)^{y^{-1}}, P^y, \hat{P}^y, P^{x_1\cdot y} \cdot X_0^{\mathsf{sk}_u} = P^{x_1\cdot y} \cdot \mathsf{pk}_u^{x_0}\right).$$

Then for $(\mathsf{sk}_{u'}, \mathsf{pk}_{u'}) \in \mathsf{UKeyGen}(\mathsf{pp})$, $\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \mathsf{sk}_{u'}, \sigma)$ outputs a new signature with the form of:

$$\sigma' = \left(\left(\prod_{i=1}^{k} C_i^{x_i}\right)^{y^{-1}}, P^y, \hat{P}^y, P^{x_1\cdot y} \cdot X_0^{\mathsf{sk}_{u'}} = P^{x_1\cdot y} \cdot \mathsf{pk}_{u'}^{x_0}\right).$$

It is clear that this looks like a fresh signature σ' in $\mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ for $\mathsf{pk}_{u'}$ with randomness y. So the output of ConvertSig is distributed the same as the output of Sign. □

Lemma D3 (Derivation-privacy) *The construction described in Section 3.3 provides Derivation-privacy.*

Proof. For all $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$, pk_u, $\boldsymbol{M}, \boldsymbol{O} = \boldsymbol{\rho}, k', k'', \boldsymbol{T}, \boldsymbol{U}$, $\mathsf{uk}_{k'}$, and σ. If $\mathsf{SC.Open}(C_j, M_j, O_j)_{j\in k} = 1 \ \wedge \ \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \boldsymbol{C}, \sigma, (\boldsymbol{T}, \boldsymbol{U})) = 1 \ \wedge \ \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$, then for an index $l = k + 1 \in [k + 1, k']$, let M_l be a message set such that the message vector is $\boldsymbol{M}^* = (\boldsymbol{M}, M_l)$ and the related commitment vector is $\boldsymbol{C}^* = (\boldsymbol{C}, C_l)$. We intend to show that ChangeRel produces outputs with the same distribution as Sign for vectors $\boldsymbol{M}^*$ and $\boldsymbol{C}^*$: $\mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}^*, k', \mathsf{pk}_u; \boldsymbol{\rho}) \approx \mathsf{ChangeRel}(M_l, \sigma, \boldsymbol{C}, \mathsf{uk}_{k'}, k'')$. More precisely, for some $y \in Z_p^*$, a signature $\sigma = (Z, Y, \hat{Y}, T) \in \mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ satisfying $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \boldsymbol{C}^*, \sigma, (\boldsymbol{T}, \boldsymbol{U})) = 1$ is of the form

$$\sigma = \left(\left(\prod_{i=1}^{k} (C_i^*)^{x_i}\right)^{\frac{1}{y}}, P^y, \hat{P}^y, P^{x_1\cdot y} \cdot \mathsf{pk}_u^{x_0}\right)$$

ConvertSig outputs the element Z of the signature as:

$$Z = \left(\prod_{i=1}^{k} (C_i^{x_i})^{\frac{1}{y}} \cdot \left(\prod_{i \in [t] \wedge l \in [k+1,k']} P^{\alpha^i \cdot x_l \cdot y^{-1}} \right)^{f_i} \right) =$$

$$\prod \left((C_i^{x_i})^{\frac{1}{y}} \cdot C_l^{x_l \cdot \frac{1}{y}} \right) = \prod (\underbrace{C_i^{x_i} \cdot C_l^{x_l}}_{\prod_{i=1}^{l} (C_i^*)^{x_i}})^{\frac{1}{y}}$$

So for the whole signature, it outputs the signature as:

$$\sigma' = \left(\left(\prod_{i=1}^{l} (C_i^*)^{x_i} \right)^{\frac{1}{y}}, P^y, \hat{P}^y, P^{x_1 \cdot y} \cdot \mathsf{pk}_u^{x_0} \right)$$

which looks like a fresh signature σ in $\mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ for $\boldsymbol{M}^*$ using the randomness y. This is, for vectors $\boldsymbol{C}^*, \boldsymbol{M}^*$, ConvertSig produces signatures with the same distribution as Sign. □

D.3 Security of DAC

Lemma D4 (Unforgeability) *Let* ZKPoK *be a ZKPoK and let* SPSEQ-UC *be unforgeable, then the* DAC *construction in Fig 3 is unforgeable.*

Proof. We show that an adversary performing an incompatible showing for a dishonest user can be used to forge an SPSEQ-UC signature. Assume a PPT adversary $\mathcal{A}$ that wins the unforgeability game (Definition D.1) with non-negligible probability and let $(\boldsymbol{C}^*, \mathsf{nym}_P^*, \mathsf{A}^*, \sigma^*)$ be the message-signature pair it uses and $\boldsymbol{W}^*$ be the witness for an attribute set $(D^*, \mathsf{A}') \notin \mathcal{R}_{k'}$ (this implies $D^* \not\subseteq \mathsf{A}'$), for all $i = \perp$ where $i, \mathsf{A}', \mathsf{dk}_{k'} \in \mathcal{L}_{\mathsf{cred}}$; moreover, the $\mathsf{ZKPoK}(\mathsf{sk}_\mathsf{P}^*, \mathsf{nym}_p^*)$ verifies. We construct an adversary $\mathcal{B}$ that breaks the unforgeability of SPSEQ-UC. We note that we extract from ZKPoK and assume this will only fail with negligible probability. Then, we are ready to reduce to the unforgeability of SPSEQ-UC:

Reduction. The reduction is straightforward. $\mathcal{B}$ interacts with a challenger $\mathcal{C}$ in the unforgeability game for SPSEQ-UC and $\mathcal{B}$ simulates the DAC-unforgeability game for $\mathcal{A}$. $\mathcal{C}$ runs $(\mathsf{pp}, \mathsf{sk}_\mathsf{CA}, \mathsf{pk}_\mathsf{CA}) \leftarrow \mathsf{Setup}(1^\lambda, 1^t, 1^\ell)$ and gives $(\mathsf{pk}_\mathsf{CA}, \mathsf{pp})$ to $\mathcal{B}$. Then, $\mathcal{B}$ sets $\mathsf{pp} = \mathsf{pp}_{\mathsf{SPSEQ\text{-}UC}}, \mathsf{vk} = \mathsf{pk}_\mathsf{CA}$ and sends them to $\mathcal{A}$. It next simulates the environment and oracles. All oracles are executed as in the real game, except for the following oracles, which use the signing oracle instead of using the signing key sk_CA:

- When the oracles $\mathcal{O}^{\mathsf{Corrupt}}$ and $\mathcal{O}^{\mathsf{User}}$ are called, $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{Corrupt}}$ and $\mathcal{O}^{\mathsf{Create}}$ of the SPSEQ-UC scheme, respectively. Note that when $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{Corrupt}}$ of the SPSEQ-UC, it gets sk_i and finally returns sk_i and all the associated credentials items to $\mathcal{A}$.
- $\mathcal{O}^{\mathsf{CreateRoot}}(i, k', \mathsf{A})$: On input a user identity i, an index k', and an attribute vector A. If $i \notin \mathcal{HU}$ it returns $\perp$, else it picks $\boldsymbol{\rho}$ for an attributes vector. Then it submits $(\mathsf{nym}_i, k', \mathsf{A}, \boldsymbol{\rho})$ to the signing oracle $\mathcal{O}^{\mathsf{Sign}}$. Receives a signature $(\sigma = (Z, Y, \hat{Y}, T), (\boldsymbol{C}, \boldsymbol{O}), \mathsf{uk}_{k'})$. It sets $\sigma = \mathsf{cred}_i$, $\mathsf{uk}_{k'} = \mathsf{dk}_{k'}$ and appends $(i, \mathsf{A}, \mathsf{dk}_{k'}, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$.
- $\mathcal{O}^{\mathsf{RootIss}}(k', \mathsf{A})$: On input an index k' and an attribute vector A. It extracts $\boldsymbol{\rho}$ from the proof of knowledge $\mathsf{ZKPoK}(\boldsymbol{\rho}, P^{\boldsymbol{\rho}})$ produced by $\mathcal{A}$ for an attributes vector. Then it submits $(\mathsf{nym}_i, k', \mathsf{A}, \boldsymbol{\rho})$ to the signing oracle $\mathcal{O}^{\mathsf{Sign}}$, where nym_i is an adversary pseudonym of a corrupted user. Receives a signature $(\sigma = (Z, Y, \hat{Y}, T), (\boldsymbol{C}, \boldsymbol{O}), \mathsf{uk}_{k'})$. It sets $(\sigma, \boldsymbol{C}, \boldsymbol{O}, \mathsf{nym}_i) = \mathsf{cred}_i, \mathsf{uk}_{k'} = \mathsf{dk}_{k'}$ and appends $(\perp, \mathsf{A}, k', \perp)$ to $\mathcal{L}_{\mathsf{cred}}$ and outputs the results.

– The oracles $(\mathcal{O}^{\mathsf{Issue}}, \mathcal{O}^{\mathsf{ObtIss}})$: In both $\mathcal{O}^{\mathsf{ObtIss}}$ and $\mathcal{O}^{\mathsf{Issue}}$, all executions of $\mathsf{ChangeRel}$ and $\mathsf{ConvertSig}$ for credentials $(i, \mathsf{dk}_{k'}, \mathsf{A}', \sigma_i) \in \mathcal{L}_{\mathsf{cred}}$ are replaced by the oracle $\mathsf{Sign}(\mathsf{sk}_{\mathsf{CA}}, \mathsf{A}', k', \mathsf{pk}_i, \boldsymbol{\rho})$, where $\mathsf{pk}_i = \mathsf{nym}_i = 1$ for the $\mathcal{O}^{\mathsf{Issue}}$ and $\mathsf{pk}_i = \mathsf{nym}_j$ for the $\mathcal{O}^{\mathsf{ObtIss}}$.

As it is clear, $\mathcal{B}$ can handle any oracle query and never aborts. So, at the end of the game, $\mathcal{B}$ simulates all oracles perfectly for $\mathcal{A}$ who is able, with some probability, to prove possession of a credential on A^*. To do this, $\mathcal{B}$ interacts with $\mathcal{A}$ as verifier in a showing protocol. If $\mathcal{A}$ outputs a valid showing proof as $(\boldsymbol{C}^*, \sigma^* = (Z^*, Y^*, \hat{Y}^*, T^*), D^*, \mathsf{nym}_p^*, \boldsymbol{W}^*)$ and conducting $\mathsf{ZKPoK}(\mathsf{sk}_{\mathsf{P}}^*, \mathsf{nym}_p^*)$ then $\mathcal{B}$ extracts from the proof of knowledge contained in the Show protocol the value sk_p^* related to the nym_P^* and stores all elements. Moreover, no credential owned by corrupt users can be valid on this set of messages D^* (as $\mathcal{A}$ can win the unforgeability game). This means that, for any credential on (sk_i) with all $i = \perp$, we have $(D^*, \mathsf{A}') \notin \mathcal{R}_{k'}$. In all cases, this means that $(\mathsf{sk}_p^*, (\boldsymbol{C}^*, D^*, \boldsymbol{W}^*), \sigma^*)$ is a valid forgery against our signature scheme, $\mathcal{B}$ breaks thus unforgeability of $\mathsf{SPSEQ\text{-}UC}$ which concludes our proof. □

Lemma D5 (Anonymity) *Let* ZKPoK *be a ZKPoK,* NIZK *be knowledge sound, the* DDH *assumption holds and the* $\mathsf{SPSEQ\text{-}UC}$ *provides Origin-hiding, Conversion-privacy and Derivation-privacy, then the* DAC *construction in Fig 3 is anonymous.*

Proof. The proof follows a sequence of games until a game where answers for the query to $\mathcal{O}_b^{\mathsf{Anon}}$ is independent of the bit b. In $\mathbf{Game}_1$ we use the knowledge soundness of NIZK to extract the signing key. Then, in $\mathbf{Game}_2$ we replace all $\mathsf{ChangeRep}$, $\mathsf{ChangeRel}$ and $\mathsf{ConvertSig}$ calls with freshly generated signatures. In $\mathbf{Game}_3$ we simulate all ZKPoKs and in $\mathbf{Game}_4$ we guess a user to be asked in $\mathcal{O}_b^{\mathsf{Anon}}$. Finally, in $\mathbf{Game}_5$ we replace the respective commitment vectors with random vectors.

$\mathbf{Game}_0$: The original game as given in Definition 9.

$\mathbf{Game}_1$: As Game_0, except when $\mathcal{A}$ outputs the $\mathsf{pk}_{\mathsf{CA}}$ and corresponding $\mathsf{NIZK}(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}})$, the experiment runs the knowledge extractor for NIZK, which extracts a witness $((x_i)_{i\in[0,\ell]}, \alpha)$ sets them as $\mathsf{sk}_{\mathsf{CA}}$ including the SC trapdoor. If the extractor fails, we abort.

$\mathsf{Game}_0 \to \mathsf{Game}_1$: The success probability in Game_1 is the same as in Game_0, unless the extractor fails, i.e., using knowledge soundness we have

$$|\Pr[S_0] - \Pr[S_1]| \leq \epsilon_{ks}(\lambda).$$

$\mathbf{Game}_2$: As Game_1, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: Like in Game_1, but for $\mu, \psi \in Z_p^*$, all executions of $\mathsf{ChangeRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\boldsymbol{C}, \boldsymbol{O}), \sigma, \mu, \psi)$ for the credential $(i_b, \mathsf{dk}_{k'}, \mathsf{A}', \sigma_b) \leftarrow \mathcal{L}_{\mathsf{cred}}[\mathsf{j}_b]$ are replaced by $\mathsf{Sign}(\mathsf{sk}_{\mathsf{CA}}, \mathsf{A}', k', \mathsf{pk}_u; \boldsymbol{\rho})$. Oracles are simulated as in Game_1 , except for the following oracles as:

– $\mathcal{O}^{\mathsf{ObtIss}}$: all executions of $\mathsf{ChangeRel}$ and $\mathsf{ConvertSig}$ for credentials $(j, \mathsf{dk}_{k'}, \mathsf{A}', \sigma_j) \in \mathcal{L}_{\mathsf{cred}}$ are replaced by $\mathsf{Sign}(\mathsf{sk}_{\mathsf{CA}}, \mathsf{A}', k', \mathsf{pk}_j, \boldsymbol{\rho})$.

$\mathsf{Game}_2 \to \mathsf{Game}_1$: By Origin-hiding, Derivation-privacy and Conversion-privacy, replacing signatures from $\mathsf{ChangeRep}$, $\mathsf{ChangeRel}$ and $\mathsf{ConvertSig}$ with ones from Sign are identically distributed for all $(\mathsf{A}, \boldsymbol{C})$. We thus have

$$\Pr[S_1] = \Pr[S_2]$$

$\mathbf{Game}_3$: As Game_2, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: All proofs $\mathsf{ZKPoK}(\mathsf{sk}_p, \mathsf{nym}_p)$ and $\mathsf{ZKPoK}(\boldsymbol{\rho}, P^{\boldsymbol{\rho}})$ in $\mathsf{CredProve}$ and $\mathsf{CreateCred}$ respectively, are simulated.

$\mathsf{Game}_2 \to \mathsf{Game}_3$: By perfect zero-knowledge of ZKPoK, we have that

$$\Pr[S_2] = \Pr[S_3] \Rightarrow \Pr[S_1] = \Pr[S_2] = \Pr[S_3]$$

$\mathbf{Game}_4$: Same as Game_3, except for the following changes. Let q_u be (an upper bound on) the number of queries made to $\mathcal{O}^{\mathsf{User}}$. At the beginning Game_4 picks $\omega \leftarrow [q_u]$ (it guesses that the user who owns the index j_b credential is registered at the ω-th call to $\mathcal{O}^{\mathsf{User}}$) and runs $\mathcal{O}^{\mathsf{User}}, \mathcal{O}^{\mathsf{Corrupt}}$ and $\mathcal{O}_b^{\mathsf{Anon}}$ as follows:

- $\mathcal{O}^{\mathsf{User}}(i)$: As in Game_3, except if this is the ω-th call to the oracle then it additionally defines $i^* \leftarrow i$.
- $\mathcal{O}^{\mathsf{Corrupt}}(i, \mathsf{pk}_u)$: If $i \in \mathcal{CU}$ or $i \in \mathcal{O}_b^{\mathsf{Anon}}$, it returns $\perp$ (as in the previous games). If $i = i^*$ then the experiment stops and outputs a random bit $b' \leftarrow \{0,1\}$. Otherwise, if $i \in \mathcal{HU}$, it returns user i's sk_i and credentials and moves i from $\mathcal{HU}$ to $\mathcal{CU}$; and if $i \notin \mathcal{HU} \cup \mathcal{CU}$, it registers and adds i to $\mathcal{CU}$ a new corrupt user with public key pk_i.
- $\mathcal{O}_b^{\mathsf{Anon}}(\mathsf{j}_0, \mathsf{j}_1, D)$: As in Game_3, except that if $i^* \neq i_b \leftarrow \mathcal{L}_{\mathsf{cred}}[\mathsf{j}_b]$, the experiment stops and outputting $b' \leftarrow \{0,1\}$.

$\mathsf{Game}_3 \rightarrow \mathsf{Game}_4$: By assumption, $\mathcal{O}_b^{\mathsf{Anon}}$ is called at least once with some input $(\mathsf{j}_0, \mathsf{j}_1, D)$ such that $i_0 \leftarrow \mathcal{L}_{\mathsf{cred}}[\mathsf{j}_0], i_1 \leftarrow \mathcal{L}_{\mathsf{cred}}[\mathsf{j}_1] \in \mathcal{HU}$. If $i^* = i_b$ then $\mathcal{O}_b^{\mathsf{Anon}}$ does not abort and neither does $\mathcal{O}^{\mathsf{Corrupt}}$ (it cannot have been called on i_b before that call to $\mathcal{O}_b^{\mathsf{Anon}}$ (otherwise $i_b \notin \mathcal{HU}$); if called afterwards, it returns $\perp$, where $i^* \in \mathcal{O}_b^{\mathsf{Anon}}$). Since $i^* = [i_b]$ with probability $\frac{1}{q_u}$, the probability thaft the experiment does not abort is at least $\frac{1}{q_u}$, and thus

$$\Pr[S_4] \geq (1 - \frac{1}{q_u})\frac{1}{2} + \frac{1}{q_u} \cdot \Pr[S_3]$$

$\mathbf{Game}_5$: As Game_4, except that for $\mathcal{O}_b^{\mathsf{Anon}}(\mathsf{j}_0, \mathsf{j}_1, D)$: it picks $\boldsymbol{C} \leftarrow (\mathbb{G}_1^*)^k$ and performs the showing using $\mathsf{cred}' \leftarrow (\boldsymbol{C}, \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}, \ldots))$, with $D = (d_i)_{i \in [k]}$ and $W_i \leftarrow f_{d_i}(a)^{-1} \cdot C_i$ for $i \in [k]$. Note that the only difference is the choice of $\boldsymbol{C}$; while $\boldsymbol{W}$ is distributed as in Game_4, in particular, they are unique elements satisfying $\mathsf{VerifySubset}(C_i, D_i, W_i)_{i \in [k]}$.

$\mathsf{Game}_4 \rightarrow \mathsf{Game}_5$: Let $(\mathsf{BG}, P^x, P^y, P^z)$ be a DDH instance (not to be confused with SPSEQ-UC elements) for $\mathsf{BG} = \mathsf{BGGen}(1^\lambda)$ where $x, y \leftarrow Z_p$ and Z is equal to $P^{x \cdot y}$ or a random element. The extended version of DDH that we consider here is given by $(P, P^{x_1}, \ldots, P^{x_k}, P^y, Z_1, \ldots, Z_k)$ where $Z_i = P^{x_i \cdot y}$ or random for all $\in \{1, \ldots, k\}$. One can easily show that this extended version of DDH follows from DDH itself (with some polynomial security loss) as long as k is a polynomial. Oracles are simulated as in Game_4, except for the following oracles as:

- $\mathcal{O}^{\mathsf{ObtIss}}(i, \mathsf{A})$: As in Game_4, except for the computation of the following values if $i = i^*$. Let this be the i-th call to this oracle. Since $\alpha \notin \mathsf{A}$, it computes $C_i \leftarrow f_{A_i}(a) \cdot P^{x_i}$ for $A_i \in \mathsf{A}$ (all C_i are thus distributed as in the original game.)
- $\mathcal{O}^{\mathsf{CredProve}}(j, D)$: As in Game_4, with the difference that if $i^* = i \leftarrow \mathcal{L}_{\mathsf{cred}}[j]$, it computes the witness $W_i \leftarrow f_{A_i/d_i}(a)^\mu \cdot P^{x_i}$. ($W_i$ is thus distributed as in the original game and $D = (d_i)_{i \in [k]}$.)
- $\mathcal{O}_b^{\mathsf{Anon}}(\mathsf{j}_0, \mathsf{j}_1, D)$: As in Game_4, with the following difference. Using self-reducibility of DDH, it picks $s, t \leftarrow Z_p^*$ and computes $Y' \leftarrow P^{t \cdot y} \cdot P^s = P^{y'}$ with $y' \leftarrow t \cdot y + s$, and $Z_i' \leftarrow P^{t \cdot z_i} \cdot P^{s \cdot x_i} = P^{(t(z_i - x_i \cdot y) + x_i \cdot y')}$. (If $z_i \neq x_i \cdot y$ then Y' and Z_i' are independently random; otherwise $Z_i' = y' \cdot X_i$) It performs the showing using the following values. Since $a \notin D$: $C_i \leftarrow f_{A_i}(a) \cdot Z_i'$ and $W_i \leftarrow f_{A_i/d_i}(a)^{-1} \cdot C_i$.

Apart from an error event happening with negligible probability, we have simulated Game_4 if the DDH instance was "real" and Game_5 otherwise. If during the simulation of $\mathcal{O}_b^{\mathsf{Anon}}$ it occurs that any $Y_i' = 0_{\mathbb{G}_1}$ or $Z_i' = 0_{\mathbb{G}_1}$ then the distribution of values is not as in one of the two games. Otherwise, we have implicitly set $\rho_i \leftarrow x_i$ and $\mu \leftarrow y'$ (for a fresh value y' at every call of $\mathcal{O}_b^{\mathsf{Anon}}$). In case of

a DDH instance, we have for all $C_i \leftarrow (P^{f_{A_i}(a)})^{\rho_i \cdot \mu}$; otherwise all C_i are independently randoms. Let $\epsilon_{\mathsf{DDH}}(\lambda)$ denote the advantage of solving the DDH problem and q_l the number of queries to the $\mathcal{O}_b^{\mathsf{Anon}}$, we have

$|\Pr[S_4] - \Pr[S_5]| \leq \epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p}$

In Game_5 the $\mathcal{O}_b^{\mathsf{Anon}}$ oracle returns a fresh signature σ on random elements $\boldsymbol{C} \leftarrow (\mathbb{G}_1^*)^k$ and a simulated proof; the bit b is thus information-theoretically hidden from $\mathcal{A}$, so we have $\Pr[S_5] = \frac{1}{2}$. From this and the above equations we have

$$\Pr[S_4] \leq \Pr[S_5] + \epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p} = \frac{1}{2} + \epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p},$$
$$\Pr[S_3] \leq \frac{1}{2} + q_u \cdot \Pr[S_4] - \frac{1}{2} \cdot q_u \leq \frac{1}{2} + q_u \cdot (\epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p}),$$
$$\Pr[S_0] \leq \Pr[S_1] + ks(\lambda) \leq \frac{1}{2} + ks(\lambda) + q_u \cdot (\epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p})$$

where $\Pr[S_1] = \Pr[S_3]$; q_u, q_o and q_l are the number of queries to the $\mathcal{O}^{\mathsf{user}}$, $\mathcal{O}^{\mathsf{Obtain}}$ and the $\mathcal{O}_b^{\mathsf{Anon}}$ oracle, respectively. Assuming security of ZKPoK, NIZK and DDH, the adversary's advantage is thus negligible. □