Practical Delegatable Anonymous Credentials From Equivalence Class Signatures

Omid Mir
Johannes Kepler University Linz
LIT Secure and Correct Systems Lab
Linz, Austria
mir@ins.jku.at

Daniel Slamanig
AIT Austrian Institute of Technology
Vienna, Austria
daniel.slamanig@ait.ac.at

Balthazar Bauer
IRIF, Université de Paris Cité
Paris, France
Balthazar.Bauer@ens.fr

René Mayrhofer
Johannes Kepler University Linz
Institute of Networks and Security
Linz, Austria
rm@ins.jku.at

ABSTRACT

Anonymous credentials (ACs) systems 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 can also be hidden.

In this paper we present a novel DAC scheme that supports attributes, provides anonymity for delegations, allows the delegators to restrict further delegations, and also comes with an efficient construction. 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, where signatures 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, the signatures and messages can be publicly randomized and thus allow unlinkable delegation and showings in the DAC system. We present further optimizations such as cross-set commitment aggregation that, in combination, enable efficient selective showing of attributes 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.

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–26

https://doi.org/XXXXXXX.XXXXXXX

KEYWORDS

Delegatable anonymous credentials, equivalence-class signatures, set commitments

1 INTRODUCTION

Anonymous credentials (ACs) [9, 10, 15, 23, 28, 30, 35, 36] 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 [19] and related concepts [33] being integrated into the Trust Tokens API[1] as well as standardized within the Internet Engineering Task Force (IETF), or the PrivateStats proposal by Facebook.[2] A recent large scale real-world application of ACs are private groups within the popular Signal messenger [14].

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. Most prominently in a public key infrastructure (PKI) there is a chain of certificates between the user certificate and a trusted root authority. A simple way to achieve delegation is credential sharing. However, giving away the full and unlimited power of a credential is often not desirable. 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 allows the secretary to sign arbitrary documents in the name of the manager – which is typically not what the manager intends. Even worse, since the actual signing key was shared, the

[1] https://web.dev/trust-tokens/
[2] https://research.fb.com/privatestats

only way to revoke these powers from the secretary is to revoke and invalidate the whole credential. This problem directly carries over to the setting of anonymous credentials and thus dedicated features for delegation are required.

Moreover, when using anonymous credentials, a party verifying a credential (e.g., a service provider) typically knows all public keys of credential issuers.[3] Especially when used in a hierarchical setting, this does not provide strong privacy guarantees [2, 7, 17]. That is, this chain of issuers (or delegators) may reveal sensitive information about the issuer's organizational structure or the credential holder. A real-world example is the current ISO mobile driving license (mDL) [29], which can, e.g., be used for age verification without disclosing other attributes — however, the specific issuer (e.g., a state DMV, to which a country delegates the capability to issue mDL credentials) implicitly reveals at least the holder's rough living location. 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. To address the aforementioned problems, Chase and Lysyanskaya [13] introduced the notion of delegatable anonymous credentials (DACs). The idea is best explained by using the notion of levels introduced by Belenkiy et al. [2]. An initial credential is issued for a level $L = 1$ and any level L credential can then be used to delegate a level $L + 1$ credential to another party. 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. Only the identity (i.e., public key) of the root issuer, but none of the intermediate levels is revealed during the verification. DACs have been meanwhile shown to have various interesting applications, among them being (physical) access control [34], root of trust [17], or authorizing transactions in permissioned blockchains [5, 7].

DAC constructions. After its introduction and theoretical treatment in [13], various more practical approaches to construct DACs have been proposed (see Section 1.4 for a more detailed comparison of practical schemes). Belenkiy et al. [2] provide the first practical treatment and in particular a construction 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 [26] resulting in credentials and showings of size linear in the chain length L. Unfortunately, while theoretically appealing, GS proofs make this scheme rather inefficient for practical use as it requires to prove expensive statements, result in poor concrete computational performance and large credential size. Several other DAC constructions have been proposed afterwards, e.g. [11, 12, 21], which follow roughly the same techniques as [2], i.e., using malleable proof systems (based on GS) as the main building block, and thus have similar performance characteristics.

Camenisch et al. [7] propose a practically efficient approach towards DAC based on a structure-preserving signature (SPS) scheme by Groth [25]. Here 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. This approach has recently been integrated into Hyperledger Fabric [5]. Later, Blömer and Bobolz (BB) [3] proposed another practical DAC approach using dynamically malleable signatures (DMS) and NIZK proofs, which conceptually is similar to the approach in [12]. A DMS can sign messages and "placeholders", which can be replaced by concrete messages during delegation and thus enabling to add attributes during delegation in the DAC.

At this point it should be mentioned that both, [7] as well as [3], require a trusted setup (and more particular a trapdoor being a decryption key in their parameters), which is an undesirable feature in privacy-preserving primitives.

DAC using equivalence-class signatures. Crites and Lysyanskaya [17] provide probably the most efficient and conceptually simplest construction of DACs, which does not require an extensive use of NIZK proofs. The main building block is a new type of signature scheme, called a *mercurial signature*, which extends structure-preserving signatures on equivalence classes (SPSEQ) [23, 27]. We recall that SPSEQ is a type of SPS for signing group element vectors $\vec{M}$. Message $\vec{M}$ can be viewed as one representative of an equivalence classes of all scaled message $\vec{M}^{\mu}$. It allows to randomize signatures and at the same time provides randomization of signed messages, i.e., to publicly switch between representatives $\vec{M}$ and $\vec{M}^{\mu}$ for any non-zero μ. The message space is required to provide an unlinkability property, which makes classes and in particular messages indistinguishable. Thus, SPSEQ allow to construct privacy-preserving primitives while in contrast to pure SPS not requiring NIZK proofs. Therefore, they allow to construct more efficient schemes. Mercurial signatures extend SPSEQ by additionally introducing equivalence classes on the keys pace and thus allow to switch (i.e., randomize) public keys and adapt signature to the switched keys. This allows to construct conceptually simple DAC schemes via chains of signed public keys.

Unfortunately, existing DAC schemes based on mercurial signatures [17, 18] have some shortcomings: *i*) They do not support attributes in a meaningful way. The first such DAC in [17] does not support attributes at all. And while Crites and Lysyanskaya in [18] show how to support attributes, it is only possible in a way such that *all* attributes are *always* revealed during showings. Thus, no selective disclosure is supported, which however is an important feature and thus prevents many privacy-preserving applications. *ii*) Bauer and Fuchsbauer [1] point out a drawback of the weak form of anonymity provided by their DAC constructions. That is, if Alice delegates a credential to Bob, she can identify Bob whenever he shows the credential, which results in a severe degradation of Bob's privacy. This lies in the nature of the way how such DACs from mercurial signatures are constructed, i.e., 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 [1] for more details). Moreover, *iii*) similar to [7], the credential size depends on the delegation chain length L, and consequently, it 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

[3] Concealing the credential issuer has recently also been shown to have value in a setting of anonymous credentials without credential delegation [4, 15].

do not provide all the desirable properties of a) supporting selective showing of attributes, b) being compact, and c) providing sufficiently strong anonymity guarantees at the same time (i.e., anonymity during both issuing and showing credentials in the presence of a corrupted CA without needing a trusted setup). Those that provide the desired features are unfortunately less efficient and/or require setup assumptions (i.e., a trusted setup).

1.1 Our Contribution

Our goal in this paper is to construct conceptually simple DACs that overcome the aforementioned problems. In particular, our contributions can be summarized as follows.

Delegatable Anonymous Credentials. We propose a novel delegatable anonymous credential 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 attributes, but only depends on the delegation depth. Second, unlike the schemes in [7, 17] the credential is not composed of a number of signatures linear in the (delegation) chain. Though, similar to [3] we need to store the randomness required to create the commitments (in their case DMS secret keys) which is linear in the 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* by deciding many further levels can be delegated and whether attributes associated to previous levels can be shown or not be shown (latter can be viewed as removing attributes). *iv*) It provides *strong anonymity* which means that it not only supports an anonymous showing phase but also provides an anonymous delegation phase. This holds under a reasonable corruption model. Anonymity even holds for maliciously generated CA's keys and it does not require a trusted setup. Finally, *v*) based on an implementation we evaluate our DAC and demonstrate 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 SPSEQ [23, 27] as well as a set commitment scheme from [23] which is used in [23] to construct anonymous credentials from SPS-EQ. Loosely speaking their idea is to use SPSEQ to sign a randomizable set commitment, where the randomization represents switching between representatives of a class. To show a credential one randomizes the set commitment and randomizes as well as adapts the signature to it. The showing then includes the randomized signature, randomized commitment, and an opening to the set commitment that reveals the attributes that should be disclosed.

Now, while in SPSEQ the message space is simply group element vectors of some bounded size ℓ, in SPSEQ-UC the message space is viewed as a vector of randomizable set commitments. Henceforth we will often just call them vectors of commitments. The key novel feature of SPSEQ-UC is that it allows to extend signed vectors by additional 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 $\vec{C}$ to a vector $\vec{C}'$ (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. As with SPS-EQ, SPSEQ-UC needs to be unlinkable, which means the same commitment–signature pair can be revealed multiple times without being linkable to each other. However, due to the added functionality the required privacy properties become more involved.

We provide a rigorous security model for SPSEQ-UC which carefully crafts privacy notions similar to SPSEQ [23] 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 [22], which is proven secure in the generic group model, and the set commitment scheme in [23]. 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 [15] (building upon and improving the SPSEQ in [31]), as an interesting avenue for future work.[4] Another direction would also be to adopt the modified set commitment scheme in [15], 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 Practical Example Application

While DAC can be beneficial in many applications of ACs, we want to discuss a particular application of DAC as a motivation for future practical deployments. As briefly mentioned above, the recently published ISO 18013-5 [29] standardizes international mobile driving licenses (mDLs). We use this as one of the most widely deployed credential use cases world-wide (as driving licenses are in practice used for many other purposes) and at the same time one of the earliest international standards for digital real-world credentials that take privacy protections into account (as, e.g., currently deployed NFC passports do not have any provisions whatsoever for privacy-sensitive use). In contrast to the previous paper/plastic card implementations, digital versions of driving licenses implement anti-forging measures through signatures by trusted issuing authorities (IAs). In particular, this "protection against forgery" relies on signatures over so-called MSOs. These mobile security objects are sets of hash commitments representing the credential

[4] However, there is an inherent issue when relying on the constructions in [15, 31]. To cope with a malicious issuer in the DAC, something that we consider, they require a trusted setup, i.e., a common reference string (CRS). Knowledge of the respective trapdoor could be exploited to break anonymity.

attributes and referred to as "issuer data authentication" in the standard, together with "protection against cloning" based on message authentication codes (MACs) on individual sessions (referred to as "mdoc authentication"). "Protection against unauthorized access" relies on consent dialogs on the holder device to allow users to select which attributes to include in a showing in combination with the provisioning of multiple MSOs[5] to provide unlinkability between different showings (as long as no identifying attributes are included) by using unique attribute hash sets and signature values in each presentation. Briefly, the current mDL standard [29] provides the following aspects of ACs: 1) selective disclosure of attributes; 2) unlinkability against verifiers, 3) including of different showings to the same verifier (under the assumption of availability of a sufficient number of MSOs for single-use showings); 4) unforgeability under the assumption of unforgeability of the signature scheme; and 5) being compact in terms of static credential/signature data and showings sizes as well as being sufficiently efficient for implementation on current smart cards and phones. However, it currently neither addresses *strong anonymity* against issuers themselves nor *delegation* of capabilities to issuing authorities[6].

Delegation is particularly important for global standards and practical deployments of such real-world credentials: it seems highly unlikely that a single issuer would be considered trustworthy for driving licenses world-wide. In many larger countries, this power is already explicitly delegated further to smaller organizational units such as US state DMVs, with delegation chains of depth at least two. However, the current standard assumes that all verifiers have access to a consolidated list of all issuer public keys and use the respective public key for verifying presented MSOs and therefore "issuer data authentication" for disclosed attributes — the implication is that the respective issuer is explicitly known to the verifier, even if no credential attribute would otherwise disclose it. This is problematic for multiple reasons, from revealing the approximate home location of a holder, potentially their citizenship, or other aspects of an internal organization of the issuing organization (e.g., if separate sub-organizations are used to issue credentials for legal aliens or asylum seekers or refugees). All of these can be abused for discrimination or other tracking.

We believe that our solution can be a future improvement for mDL or similar real-world credentials, as it explicitly supports stronger anonymity against issuers and verifiers and delegation with potential restrictions on levels (top-level issuers can decide how deep their respective organizational hierarchy is). In contrast to some other proposals, it still supports selective attribute disclosure and is compact and efficient in all messages and processing. Specifically for widely applicable use cases like age verification, one of the use-cases of mDLs, preventing this unintended leakage of personal data through the delegation hierarchy could significantly improve user privacy and prevent potential discrimination.

[5]Note that it is at the discretion of an issuing authority to provide multiple MSOs for a provisioned mDL. If the IA does not provide them, holders of mDLs remain trivially linkable between different showings based on the MSO.

[6]The current mDL standard also doesn't address efficient revocation of credentials, but recommends short-lived MSO signatures as a mitigation. We consider revocation to be out of scope here and therefore don't point it out as another shortcoming.

1.3 High Level Idea of Our Approach

On a very high level, our approach to construct DAC takes inspiration from the anonymous credentials in [23] as well as the approach based on DMS in [3]. Importantly, in contrast to the latter and similar to DACs based on mercurial signatures [17, 18], it however 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 commitment.[7] The commitment carries the attributes for the first delegatee and the public key to which the signature is tied is the one of the delegatee. The delegatee, if provided with a corresponding update key for this signature by the delegator, can then perform further delegations. This update key allows to further extend the commitment vector (and thus add attributes) 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. Due to the privacy properties of the SPSEQ-UC, a signature resulting from extending the vector looks like a fresh signature (derivation-privacy) and one resulting from switching a user key also looks like a fresh signatures (conversion-privacy). This ensures that in the DAC, delegations cannot be tracked and all credentials in a delegation chain are indistinguishable. 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. Due to the origin-hiding property of the SPSEQ-UC this results in an unlinkable showing. As we show in Section 3.4 we can realize a cross-commitment aggregation technique to make the opening of multiple commitments compact.

1.4 Comparison with Previous Work

In Table 1, we provide a comparison of our approach with other existing efficient DAC schemes in the literature [3, 7, 17, 18]. We compare our DAC with these schemes in terms of the following criteria: **Attr** indicates whether credentials include attributes that can at least be selectively revealed. We use $\approx$ to indicate that [18] supports attributes, but as all attributes always need to be revealed it does not support selective disclosure. **Expr** represents 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, e.g., ISO 18013-5 [29] discussed in Section 1.2. **Rest** indicates whether it is possible to apply a re-

[7]Technically to guarantee anonymity we require two commitments, where the first one is a dummy commitment and not assigned any or simply some fixed attributes.

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

Scheme	Attr	Expr	Rest	SAnon	|Cred|	|Show|
BB [3]	✓	S/R	≈	◐†	$O(1)$	$O(u)$
CDD [7]	✓	S/R	×	◐†,♣	$O(nL)$	$O(uL)$
CL [18]	≈	×	×	◐*	$O(nL)$	$O(uL)$
Ours	✓	S	✓	●‡	$O(1)$	$O(L)$

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

striction 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 "unshowable" by not providing the opening of the respective commitments, and, iii) how many attributes in each level (commitment) can be delegated by determining (potentially removing) key components in $\mathsf{uk}_{k'}$. Here, we note that BB [3] 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 cannot 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 under a malicious key generation by a (corrupted) root authority, and user's key can leak [8]. Here ● means that the scheme satisfies all conditions, and ◐ means that it does not provide one or more of them.

With **|Cred|** we denote the size of the credential. L indicates the length of the delegation chain. As it turns out, BB [3] (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 efficient as it needs only a constant number of group elements (5 elements) and the commitment vector with the size of delegation L. BB does not have the concept of levels and needs to send the signature (2 elements) and the elements of proving knowledge of undisclosed attributes ($|\mathsf{ZKPoK}| = u$). Note that for practical use-cases, we can typically assume $L < u$. For other schemes, this cost is much higher as their credentials grow linearly in the number of attributes and L. Note that if there are a large number of attributes to be issued, they can be split into two sets and embedded in two credentials.

Comparing efficiency with related work. We do not provide a comparison with CL as it does not support selective showing of attributes (and there is also no implementation available). CDD is the most efficient scheme and the only one from Table 1 that is fully specified. We will compare our implementation to the implementation of CDD [7] that has recently been provided by [5] in Section 5. Moreover, we provide a more comprehensive theoretical comparison with CDD, which due to the lack of space is deferred to Appendix C. For BB [3], unfortunately, only the underlying signature scheme based on Pointcheval-Sanders signatures [35] is specified. But the remaining parts of their generic constructions are not detailed, making concrete performance estimates hard. However, since their credential 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.

[8] We note that CL and our model require a credential cred_b of the anonymity challenge to be 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.

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 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 $\boldsymbol{O}$ 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 $\vec{v} = (v_1, \ldots, v_n)$ to denote a vector. Given two vectors $\vec{v}$ and $\vec{w}$, we write $(\vec{v}, \vec{w})$ for appending $\vec{w}$ to $\vec{v}$. Whenever we have vectors $\vec{w}$ and $\vec{v}$ of identical dimension whose components are sets, then by $\vec{v} \subseteq \vec{w}$ we mean that the relation is applied componentwise.

2.1 Set Commitments

Fuchsbauer et al. in [23] introduced the notion of a set commitment SC with subset openings. SC allows committing to a set $S \subset \mathbb{Z}_p$ 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 recall their construction below. We thereby make one property that is satisfied by the set commitment construction in [23] 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 as well as randomness ρ and μ we have that:

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

Set Commitment Construction of [23]. To simplify our description, we ignore the case that a set S contains the trapdoor α. For a non-empty set S, [23] defines the polynomial $f_S(X) := \prod_{s \in S}(X -$

$s) = \sum_{i=0}^{|S|} f_i \cdot X^i$. 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 $\left(P^{\alpha^i}\right)_{i=0}^{|S|}$:

- SC.Setup$(1^\lambda, 1^t) \to \mathsf{pp}_{\mathsf{sc}}$: 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 $\alpha \leftarrow \mathbb{Z}_p$ and output $\mathsf{pp}_{\mathsf{sc}} \leftarrow (\mathsf{BG}, (P^{\alpha^i}, \hat{P}^{\alpha^i})_{i\in[t]})$, which defines message space $S_{\mathsf{SC}} = \{S \subset \mathbb{Z}_p | 0 < |S| \leq t\}$. $\mathsf{pp}_{\mathsf{sc}}$ will be an implicit input to all algorithms.
- SC.Commit$(S) \to (C, O)$: On input a set $S \in S_{\mathsf{SC}}$: pick $\rho \leftarrow \mathbb{Z}_p^*$, compute $C \leftarrow (P^{f_S(\alpha)})^\rho \in \mathbb{G}_1^*$ and output (C, O) with $O \leftarrow \rho$.
- 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 $\rho \notin \mathbb{Z}_p^*$ or $S \notin S_{\mathsf{SC}}$ then return $\perp$. Otherwise if $O = \rho$ and $C = (P^{f_S(\alpha)})^\rho$, return 1; else return 0.
- 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 $0 \leftarrow$ SC.Open(C, S, O) or $T \not\subseteq S$ or $T = \emptyset$ then return $\perp$. If $O = \rho$, output $W \leftarrow (P^{f_{S\setminus T}(\alpha)})^\rho$.
- 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 $T \notin S_{\mathsf{SC}}$, return 0. Else if $W \in \mathbb{G}_1^* \wedge e(W, \hat{P}^{f_T(\alpha)}) = e(C, \hat{P})$, return 1; else 0.
- SC.RndmzC$(C, O, \mu) \to (C', O')$: On input a set commitment C, an opening information O and a randomness $\mu \in \mathbb{Z}_p$, output $C' = C^\mu$ and $O' = \mu \cdot O$.

On computing commitments. With the knowledge of 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 SC.Commit(S, α, P^ρ), 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: *i*) 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. *ii*) SPSEQ-UC not only considers signing representatives of classes of a single projective equivalence relation $\mathcal{R}$, but a family of relations. This is, 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 technically, 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 adapting a signature on a commitment vector $\vec{C}$ in class $[\vec{C}]_{\mathcal{R}^k}$ to another representative of the given class, one also can update a commitment vector $\vec{C}$ (i.e., extending it) to a vector $\vec{C}'$ being in a class $[\vec{C}']_{\mathcal{R}^{k'}}$ of a new equivalence relation. Then one can adapt the signature accordingly to the updated commitment vector. Finally, *iii*) in SPSEQ-UC a signature is bound to a user public key. The signer produces a signature bound to a user public key, and this 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 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 representing set commitments (a commitment vector) from $(\mathbb{G}_1^*)^\ell$. As mentioned above since we require updating, i.e., extending, the commitment vector, in contrast to SPSEQ, we consider a family of equivalence relations $\mathbb{R}^\ell$. Thus, for any k with $1 < k \leq \ell$, we can define the following equivalence relation $\mathcal{R}^k \in \mathbb{R}^\ell$ and the equivalence class $[\vec{C}]_{\mathcal{R}^k}$ of a set commitment vector $\vec{C} = (C_1, \ldots, C_k)$. More concretely, for a fixed bilinear group BG and (k, ℓ), we define $\mathcal{R}^k \in \mathbb{R}^\ell$ as follows:

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

Now we are ready to present the definition.

*Definition 3.1 (*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:

- PPGen$(1^\lambda, 1^t, 1^\ell) \to (\mathsf{pp})$: 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_{SC} is well-defined from pp. pp will be an implicit input to all algorithms.
- KeyGen$(\mathsf{pp}) \to (\mathsf{vk}, \mathsf{sk})$: On input the public parameters pp, this probabilistic algorithm outputs a verification and signing key pair $(\mathsf{vk}, \mathsf{sk})$.
- UKeyGen$(\mathsf{pp}) \to (\mathsf{sk}_u, \mathsf{pk}_u)$: On input the public parameters pp, this probabilistic algorithm outputs a key pair $(\mathsf{sk}_u, \mathsf{pk}_u)$ for a user u.
- RndmzC$(\vec{C}, \vec{O}, \mu) \to (\vec{C}', \vec{O}')$: This deterministic algorithm takes as input a commitment vector $\vec{C}$ of size $1 < k \leq \ell$, corresponding openings $\vec{O}$ and randomness μ. It runs $(C_i', O_i') \leftarrow$ SC.RndmzC(C_i, O_i, μ) for all $i \in [k]$ and outputs a new representative of the set commitment vector $\vec{C}' \in [\vec{C}]_{\mathcal{R}^k}$ and corresponding openings $\vec{O}'$.
- Sign$(\mathsf{sk}, \vec{M}, k', \mathsf{pk}_u; \vec{\rho}) \to (\sigma, (\vec{C}, \vec{O}), \mathsf{uk}_{k'})$: This probabilistic algorithm takes as input a signing key sk, a vector of set messages $\vec{M} = (M_1, \ldots, M_k)$, an index k' with $k \leq k' \leq \ell$, a user public key pk_u and a vector of randomness $\vec{\rho}$. It computes $(C_j, O_j)_{j\in[k]} \leftarrow$ SC.Commit$(M_j; \rho_j)$ for all $j \in [k]$, sets $\vec{C} = (C_1, \ldots, C_k)$ and $\vec{O} = (O_1, \ldots, O_k)$. It outputs a signature $(\sigma, \vec{C})$ for pk_u, and also an update key $\mathsf{uk}_{k'}$ in case $k' \neq \ell$.
- Verify$(\mathsf{vk}, \mathsf{pk}_u, \vec{C}, \sigma, (\vec{T}, \vec{U})) \to 0/1$: On input a verification key vk, a user public key pk_u, a commitment vector $\vec{C} = (C_1, \ldots, C_k)$, the purported signature σ, and a pair (subset/witness form set commitments) $(\vec{T}, \vec{U})$, it outputs 0 if any of the following checks fail and 1 otherwise:

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

- Check whether σ is a valid signature for $(\vec{C}, \mathsf{pk}_u)$.
- For all $(T_i, U_i) \in (\vec{T}, \vec{U})$: if $U_i = W_i$ check $1 = \mathsf{SC.VerifySubset}(C_i, T_i, W_i)$. Else if $U_i = O_i$, check $1 = \mathsf{SC.Open}(C_i, T_i, O_i)$.

$\mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) \to 0/1$: On input a verification key vk, an update key $\mathsf{uk}_{k'}$, an integer k' and a signature σ, this update key verification algorithm outputs 0 or 1.

$\mathsf{RndmzPK}(\mathsf{pk}_u, \psi, \chi) \to \mathsf{pk}'_u$: On input a user public key pk_u and randomness ψ, χ, this public key randomization algorithm outputs the randomized public key pk'_u.

$\mathsf{ChangeRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\vec{C}, \vec{O}), \sigma, \mu, \psi) \to (\sigma', (\vec{C}', \vec{O}'), (\mathsf{uk}'_{k'}$ or $\perp$ $), \mathsf{pk}'_u, \chi)$: This algorithm takes as input the user public key pk_u, a commitment vector $\vec{C} = (C_1, \ldots, C_k)$ in equivalence class $[\vec{C}]_{\mathcal{R}^k}$ and corresponding openings $\vec{O}$, a signature σ for $\vec{C}$, 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 $(\vec{C}', \vec{O}') \leftarrow \mathsf{RndmzC}(\vec{C}, \vec{O}, \mu)$ such that $\vec{C}' \in [\vec{C}]_{\mathcal{R}^k}$ 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 $\mathsf{uk}'_{k'}$.

$\mathsf{ChangeRel}(M_l, \sigma, \vec{C}, \mathsf{uk}_{k'}, k'') \to (\sigma', (\vec{C}', O_l), \mathsf{uk}_{k''})$: On input a message set $M_l \subset S_{\mathsf{SC}}$ for $l = k + 1 \in [k']$, a signature σ for a vector of commitments representative $\vec{C} = (C_1, \ldots, C_k)$ of equivalence class $[\vec{C}]_{\mathcal{R}^k}$, an updatable key $\mathsf{uk}_{k'}$, and an index $k'' \leq k'$. This algorithm adapts a signature σ' for a new commitment vector $\vec{C}' = (\vec{C}, C_l)$ of equivalence class $[\vec{C}']_{\mathcal{R}^l}$, 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 $\mathsf{uk}_{k''}$.

$\mathsf{SendConvertSig}(\mathsf{vk}, \mathsf{sk}_u, \sigma) \to (\sigma_{\mathrm{orph}})$: 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}.

$\mathsf{ReceiveConvertSig}(\mathsf{vk}, \mathsf{sk}_{u'}, \sigma_{\mathrm{orph}}) \to \sigma'$: 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.

We note that UKVerify is an algorithm that checks whether the update key is formed correctly. This is required in the DAC construction (cf. Section 4.2) to verify whether a delegation is valid. Moreover, it helps in the definition of the privacy properties below.

3.2 Security Definitions

Similar to conventional signatures, a SPSEQ-UC scheme needs to be correct and unforgeable. And similar to SPSEQ, we need additional properties covering the distribution of adapted signatures.

Correctness. We require that honest signatures verify as expected. Moreover, algorithms ConvertSig, ChangeRep and ChangeRel need to output valid signatures for the respective parameters. We provide a formal correctness definition in Appendix D.1.

Unforgeability. Here, 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. Here 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 is required to make this concept useful in the application to DACs. Looking ahead, since the output of ChangeRel and ChangeRep is distributed identical to Sign, we do not need 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 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 ConvertSig and ChangeRep are not considered as forgeries.

Definition 3.2. Let k, ℓ be integers. For any $\ell \geq k' > k$, we define the relation $\mathcal{R}_{k'}$ for two vectors $\vec{M} = (M_1, \ldots, M_k)$ and $\vec{M}^* = \left(M_1^*, \ldots, M_{k'}^*\right)$ as follows:

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

Formally for unforgeability we require the following:

Definition 3.3 (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 three privacy properties that are similar in vein to origin-hiding and signature adaption from previous works on SPSEQ and mercurial signatures [17, 23]. However, since SPSEQ-UC supports more functionality we need to introduce additional notions. Firstly we adapt *origin-hiding* from [17, 23] to SPSEQ-UC. We want to guarantee that fresh and randomized signatures are indistinguishable, which is important to guarantee the anonymity of showings in DACs. Secondly, we newly introduce *derivation-privacy*, which guarantees that signatures obtained by extending commitment vectors are indistinguishable from fresh signatures on extended commitment vectors. Thirdly, with introduce *conversion-privacy*, which guarantees that when switching a user key in the signature, the resulting signature is indistinguishable from a fresh signature on the new user public key. We note that these properties compose (the outputs are always valid signatures and update keys) and can thus be applied an arbitrary number of times and in an arbitrary order. The notions are essential to the anonymity (of delegation) in the DAC application. Subsequently, we use $\approx$ to denote perfect indistinguishability.

Origin-hiding (also called signature adaptation [22, 23]) formalizes the fact that signatures for well-formed commitment vectors and well-formed update keys output by ChangeRep are distributed identical to fresh signatures on the new representative.

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

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

  return:

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

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

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

  return pk_u

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

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

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

- 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** $\perp$

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

Definition 3.4 (Origin-hiding). For all (λ, t, ℓ) and $\mathsf{pp} \in \mathsf{PPGen}(1^\lambda, 1^t, 1^\ell)$, for all vk, pk_u, $\vec{C}$, $\vec{M}$, $\vec{O}$, $\vec{T}$, $\vec{U}$, $\mathsf{uk}_{k'}$, k' and σ. If $\mathsf{pk}_u \in \mathsf{UKeyGen}$ and $\mathsf{SC.Open}(\mathsf{pp}, C_j, M_j, O_j)_{j\in k} = 1 \wedge \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \vec{C}, \sigma, (\vec{T}, \vec{U})) = 1$, then for all μ, ψ, the algorithm $\mathsf{ChangeRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\vec{C}, \vec{O}), \sigma, \mu, \psi)$ outputs a uniformly random $\vec{C}' \in [\vec{C}]_{\mathcal{R}^k}$ and uniformly random pk'_u, and σ' (and if $\mathsf{uk}_{k'} \neq \perp$ update key $\mathsf{uk}'_{k'}$) in the respective spaces.

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

Definition 3.5 (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, $\vec{M}$, $\vec{O} = \vec{\rho}$, $\vec{T}$, $\vec{U}$, $\mathsf{uk}_{k'}$, k', and σ. If $\mathsf{pk}_u \in \mathsf{UKeyGen}$ and $\mathsf{SC.Open}(\mathsf{pp}, C_j, M_j, O_j)_{j\in k} = 1 \wedge \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \vec{C}, \sigma, (\vec{T}, \vec{U})) = 1 \wedge \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$, then, for all $k'' \in [k+1, k']$, M_l, we have $(\sigma', (\vec{C}', O_l), \mathsf{uk}_{k''}) \leftarrow \mathsf{ChangeRel}(M_l, \sigma, \vec{C}, \mathsf{uk}_{k'}, k'')$ and the following holds:

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

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

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

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

Remark 1. *We can also define a class-hiding notion in the vein of [17, 23] (cf. Appendix D.2). However, analogous to [22] (Proposition 1), the origin-hiding notion together with the indistinguishability of the message space (under DDH) implies 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 scheme.*

3.3 Construction

We now present our SPSEQ-UC construction and start with an intuition behind our approach. We start from the SPSEQ scheme in [22]. Inspired by their implicit use of SC in [23] 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 $M_i \subset \mathbb{Z}_p^t$ into a set commitment C_i and signs a commitment vector $(\mathbb{G}_1^*)^k$ of size $k \leq \ell$. The randomization of the 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 the commitment vector 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 feature of updating a commitment vector by appending new commitments, and the support to adapt a signature to this updated commitment vector. Therefore, 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 add a new commitment (message set) to the signed commitment vector using algorithm ChangeRel that receives a new message set, a signature and the update key. It first encodes this set to the a 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 the updated commitment vector. For the user's public key bound to a signature, instead of signing the user's public key by including it in the vector, we define the extra element T to tie the signature to this 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 $\alpha \leftarrow \mathbb{Z}_p^*$ 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 $S_{\mathsf{SC}} \leftarrow \{M \subset \mathbb{Z}_p | 0 < |M| \le t\}$. Output $\mathsf{pp} = \{\mathsf{BG}, \mathsf{pp}_{\mathsf{SC}}, S_{\mathsf{SC}}, \ell\}$.

$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{vk}, \mathsf{sk})$: For $0 \le i \le \ell$ pick $x_i \leftarrow (\mathbb{Z}_p^*)^\ell$, 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 $0 \le i \le \ell$. Output $(\mathsf{vk}, \mathsf{sk})$.

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

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

$\mathsf{Sign}(\mathsf{sk}, \vec{M}, k', \mathsf{pk}_u; \vec{\rho}) \to (\sigma, (\vec{C}, \vec{O}), \mathsf{uk}_{k'})$: On input the signing key sk, a vector of message sets $\vec{M} = (M_1, \ldots, M_k)$, an index k', $k \le k' \le \ell$, a user public key pk_u and a vector of randomness $\vec{\rho}$. 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 $\vec{C} = (C_1, \ldots, C_k)$ related to a vector of sets $\vec{M}$ and opening $\vec{O} = (O_1, \ldots, O_k)$. More precisely, SC.Commit computes a set commitment for each M_j in $\vec{M}$ 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 $\rho_j \in \vec{\rho}$, 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 elements in pp. Then, compute a signature σ for $(\mathsf{pk}_u, \vec{C})$ as follows: Pick a random $y \leftarrow \mathbb{Z}_p^*$ 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 $k \ne \ell$, 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 $(\sigma, (\vec{C}, \vec{O}), \mathsf{uk}_{k'})$.

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

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

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

- For all $(T_i, U_i) \in (\vec{T}, \vec{U})$ 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:

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

Else $U_i = O_i$, then $T_i = M_i$ is a message set and O_i is 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 $\sigma = (Z, Y, \hat{Y}, T)$, parse $\mathsf{uk}_{k'} = (\mathsf{usign}_j = ((P^{\alpha^i})^{x_j})^{y^{-1}})_{j\in[k+1,k'],i\in[t]}$ output 1 if the following holds; else 0:

$$\bigwedge_{i\in[t],j\in[k+1,k']} e(P^{\alpha^i}, \hat{X}_j) = e(((P^{\alpha^i})^{x_j})^{y^{-1}}, \hat{Y})$$

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

$\mathsf{ChangeRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\vec{C}, \vec{O}), \sigma, \mu, \psi) \to (\sigma', (\vec{C}', \vec{O}'), (\mathsf{uk}'_{k'} \text{ or } \perp), \mathsf{pk}'_u, \chi)$: On input a user public key pk_u, optionally an update key $\mathsf{uk}_{k'}$, a commitment vector $\vec{C} = (C_1, \ldots, C_k)$ in equivalence class $[\vec{C}]_{\mathcal{R}^k}$ with corresponding openings $\vec{O}$, a valid signature σ for $\vec{C}$ and randomness $\psi, \mu \in \mathbb{Z}_p^*$.
Pick $\chi \leftarrow \mathbb{Z}_p^*$ and compute a new commitment representative $(\vec{C}', \vec{O}') \leftarrow \mathsf{RndmzC}(\vec{C}, \vec{O}, \mu)$ 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 $\mathsf{uk}_{k'} \ne \perp$, check if $\mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$, randomize the update key:

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

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

$\mathsf{ChangeRel}(M_l, \sigma, \vec{C}, \mathsf{uk}_{k'}, k'') \to (\sigma', (\vec{C}', O_l), \mathsf{uk}_{k''})$: On input a message set $M_l \subset S_{\mathsf{SC}}$ for $l = k + 1 \in [k']$, a valid signature σ for commitment vector $\vec{C} = (C_1, \ldots, C_k)$ in equivalence class $[\vec{C}]_{\mathcal{R}^k}$, a valid update key $\mathsf{uk}_{k'}$, and an index $k'' \le k'$. 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 SC.Commit, but using keys of the l-component of $\mathsf{uk}_{k'}$ as

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

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

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

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

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

SendConvertSig$(\mathsf{vk}, \mathsf{sk}_u, \sigma) \to \sigma_{\text{orph}}$: 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 $\sigma_{\text{orph}} = \left(Z, Y, \hat{Y}, T' = T \cdot (X_0^{\mathsf{sk}_u})^{-1}\right)$

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

The proof of Theorem 3.7 is provided in Appendix E.3.

Theorem 3.8 (Privacy). *Our* SPSEQ-UC *construction is origin-hiding, provides conversion-privacy and derivation privacy as specified in definitions 3.4, 3.5 and 3.6, respectively.*

The proof of Theorem 3.8 is provided in Appendix E.2.

3.4 Cross-Set Commitment Aggregation

Now we introduce an aggregatable set commitment CSCA, that allows non-interactive aggregation of witnesses across multiple commitments. We use a technique inspired by [6, 24] 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 π. 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. Additionally, we add a version of the commit algorithm (used in the construction in Fig 2) that takes the commitment randomness in a point (analogously to what we have done in Section 2.1):

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

CSCA.VerifyAcross$(\{C_j, T_j\}_{j\in[k]}, \pi) \to b$. On 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.

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

Construction. Due to space reasons, we provide our new CSCA construction including all algorithms in Appendix B. We require correctness of opening as in Section 2.1, but extend it to cross-commitment aggregation in a natural way. We note that in the DAC application 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 significantly and improves verification efficiency by saving k pairings. Note that without this aggregation, the verification has the form: $\prod_{i\in[k]} e(W_i, \hat{P}^{f_{T_i}(\alpha)}) = \prod_{i\in[k]} e(C_i, \hat{P})$.

4 DELEGATABLE ANONYMOUS CREDENTIALS

We now present our definition of delegatable anonymous credentials. It is similar to [17, 23], but splits up the issuing protocol for issuing a root credential (CreateCred) and delegating a credential (IssueCred) and works as follows: A root issuer (called CA) issues a level-1 ($L = 1$) credential (a root credential) to intermediate issuers using the CreateCred protocol. The credential is assigned for a user sk_u (whom it knows with a pseudonym nym_u) with an attribute vector A and the related set commitments $\vec{C}$ rooted at $\mathsf{pk}_{\mathsf{CA}}$. CA also creates a delegation key $\mathsf{dk}_{L'}$ which determines how the credential can be delegated further until level L'. 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 switching to R's public key and possibly adding another set of attributes A′. Showing the credential consists of the user proving possession of the secret key sk_u and providing a randomized signature with required attributes.

Definition 4.1 (Delegatable anonymous credentials). DAC includes algorithms (Setup, KeyGen, NymGen) and protocols CreateCred/ReceiveCred, IssueCred/ReceiveCred for issuing a credential and CredProve/CredVerify for showing of credentials as:

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 depth (i.e., level) 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 implicitly input to all algorithms.

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).

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 (randomness) needed to use nym.

Issuing a root credential:

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

Issuing/delegating a credential:

[IssueCred$(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{L'}, \mathsf{sk}_\mathsf{u}, \mathsf{cred}_\mathsf{u}, A_l, L'') \leftrightarrow$ ReceiveCred$(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_\mathsf{r}, A_l)] \to (\mathsf{cred}_\mathsf{r}, \mathsf{dk}'_{L''})$: It is an interactive protocol between an issuer who is known by nym_u and runs the IssueCred algorithm, and a receiver, who is known by nym_r and runs the ReceiveCred side. The common inputs are the pp, CA 's public

key $\mathsf{pk}_{\mathsf{CA}}$, and attribute set A_I. Also, the issuer takes as input the issuer's secret key sk_{u} and his own credential $\mathsf{cred}_{\mathsf{u}}$ (a signature) together with all information associated with it (e.g. A, the delegatable key $\mathsf{dk}_{L'}$, the pseudonym $\mathsf{nym}_{\mathsf{u}}$ and its associated auxiliary information (randomness) aux_u), and (optionally) a level $L'' < L'$ (if not set $L'' := L'$). 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 $\mathsf{A}' = (\mathsf{A}, A_I)$ and $\mathsf{dk}'_{L''}$ (if further delegation is allowed) or $\perp$.

Showing of a credential:

$[\mathsf{CredProve}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{nym}_{\mathsf{P}}, \mathsf{aux}_{\mathsf{P}}, \mathsf{cred}_{\mathsf{P}}, D) \leftrightarrow \mathsf{CredVerify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{nym}_{\mathsf{P}}, D)] \rightarrow (0, 1)$: 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 CredVerify side. The common inputs are the pp, the CA 's public key $\mathsf{pk}_{\mathsf{CA}}$, and subset attributes that needed to be disclosed D. 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 associated 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 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 4.2 (Correctness of DAC*).* DAC is correct if Setup, KeyGen, NymGen, CreateCred, and issuing/receiving protocols are executed correctly on honestly generated inputs, then in an honest execution of the proving/verifying protocol, the verifier will accept the credential cred with probability 1.

4.1 Security of DAC

We define our security model based on the game-based framework in [23], with some modifications to harmonize their definition with DAC. $\mathcal{A}$ has access to oracles that describe the adversary's power and the possible ways to interact with the system. Briefly, $\mathcal{A}$ can perform the following actions: corrupt users using O^{Corrupt}, obtain a root credential from CA using O^{RootIss}, corrupt a CA and issue a root credential using O^{RootObt}, request credentials from a delegator using O^{Issue}, corrupt a delegator and issue a credential using O^{Obtain}, and interact with provers in showing protocol by $O^{\mathsf{CredProve}}$. Due to space constraints, we defer a detailed description of the oracles to Appendix D.3.

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.

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 for.

We provide a more comprehensive discussion and formally define the unforgeability and anonymity games in Appendix D.3.

4.2 Construction of DAC

We first give an intuition of our construction. To obtain a root credential from a CA, a user needs to send one of his pseudonyms to the CA and use some mechanism to authenticate with the real identity. Latter is outside of the protocol and omitted here. The initial nym is viewed as a user public key pk in the SPSEQ-UC scheme Σ, one can always drive a new nym using NymGen, which calls RndmzCpk of Σ to randomize the nym, and the respective signature can be adapted to the randomized nym. CA then creates a root credential (signature of Σ) on a vector of two commitments and a delegation key. Regarding these first two commitments in CreateCred protocol, we can assume the first dummy commitment for a fixed set A_1 (that is used by all credentials and are never shown in practice) and the second commitment holds the initial attribute set A_2. We only require this restriction for the root credential.

Via the IssueCred protocol, users U can then delegate their credentials to another user R using the delegatable key. U needs to derive a signature on R's secret key sk_{r} without being given the key directly. This is achieved by removing U's pseudonym $\mathsf{nym}_{\mathsf{u}}$ and switching to R's pseudonym $\mathsf{nym}_{\mathsf{r}}$ using ConvertSig of Σ and adding any additional attribute set A_i using ChangeRel of Σ. To show a credential, a user U randomizes the credential σ_{u} and $\mathsf{nym}_{\mathsf{u}}$ using ChangeRep of Σ and providing a ZKPoK that demonstrates that the secret key and randomness $\mathsf{sk}_{\mathsf{u}}, \mathsf{aux}_u$ belong to the new (randomized) $\mathsf{nym}_{\mathsf{u}}$ along with opening subsets of the attributes D using the CSCA scheme.

We provide our DAC construction in Fig 2. It is based on our SPSEQ-UC signature (denoted by Σ) from Section 3.3 and the CSCA scheme from Section 3.4. We use the following notation: Assume attribute universe $\mathcal{U} = S_{\mathsf{SC}}$, $\mathsf{A} = \vec{M}$, the updatable key as a delegatable key $\mathsf{uk}_{k'} = \mathsf{dk}_{L'}$ and $k' = L' + 1$ and the root authority's public key $\mathsf{pk}_{\mathsf{CA}} = \mathsf{vk}$. Then each credential is parameterized with a vector $\mathsf{A} = (A_1, \ldots, A_k)$, where $A_i \subseteq \mathcal{U}$ is an attribute set and consider the relation in Definition 3.2 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 a (non)-interactive zero-knowledge proof of knowledge of witness w for statement x and $\mathsf{ZKPoK}(w, x) = 1$ if verifier accepts (cf. Appendix A.2 for a formal treatment). In Fig 2, we replace SC by CSCA (cf. Section 3.4 and its construction in Appendix B) to improve the communication bandwidth by aggregating witnesses and also verification efficiency due to batching of pairing equations. It also allows creating a set commitment for openings $\vec{\rho}$ in the group as $P^{\vec{\rho}}$. We stress that CSCA is fully compatible with SC and provides identical functionality.

Theorem 4.3. *The* DAC *construction in Fig 2 is correct, unforgeable, and anonymous.*

The proof, which is based on our core SPSEQ-UC using SC, is presented in Appendix E.1. Unforgeability is essentially based on the unforgeability of SPSEQ-UC and anonymity essentially follows from the privacy properties of SPSEQ-UC and the DDH assumption.

4.3 Discussion of Various Aspects

We also discuss various aspects of our DAC construction when it comes to implementation and supported features.

$\mathsf{Setup}(1^\lambda, 1^t, 1^\ell)$: Pick $\alpha \leftarrow \mathbb{Z}_p$ and run $\mathsf{pp}_\Sigma \leftarrow \Sigma.\mathsf{Setup}(1^\lambda, 1^t, 1^{(\ell+1)}; \alpha)$. 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 $\pi_{\mathsf{CA}} \leftarrow \mathsf{NIZK}(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}})$. Output $\mathsf{pp} = (\mathsf{pp}_\Sigma, \mathsf{pk}_{\mathsf{CA}}, \pi_{\mathsf{CA}})$. The attribute space is $\mathcal{U} = S_{\mathsf{pp}_\Sigma}$ of pp_Σ.

$\mathsf{KeyGen}(\mathsf{pp})$: Pick $w_{\mathsf{u}} \leftarrow \mathbb{Z}_p^*$, set $\mathsf{pk}_{\mathsf{u}} \leftarrow P^{w_{\mathsf{u}}}$ and $\mathsf{sk}_{\mathsf{u}} = w_{\mathsf{u}}$, and return $(\mathsf{sk}_{\mathsf{u}}, \mathsf{pk}_{\mathsf{u}})$.

$\mathsf{NymGen}(\mathsf{pk}_{\mathsf{u}})$: Pick randomness $\psi, \chi \leftarrow \mathbb{Z}_p^*$, compute $\mathsf{nym}_{\mathsf{u}} = \mathsf{pk}'_{\mathsf{u}} \leftarrow \Sigma.\mathsf{RndmzPK}(\mathsf{pk}_{\mathsf{u}}, \psi, \chi)$, set $\mathsf{aux}_{\mathsf{u}} = (\chi, \psi)$, and output $(\mathsf{nym}_{\mathsf{u}}, \mathsf{aux}_{\mathsf{u}})$.

Issuing a root credential $[\mathsf{CreateCred}(L', \mathsf{A}, \mathsf{sk}_{\mathsf{CA}}) \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{u}}, \mathsf{A})] \rightarrow (\mathsf{cred}, (\vec{C}, \vec{O}), \mathsf{dk}_{L'})$:

U picks $\rho_j \leftarrow \mathbb{Z}_p^*$, sends $((P^{\rho_j})_{j\in[2]}, \mathsf{nym}_{\mathsf{u}}, \mathsf{A} = (A_1, A_2))$, to CA and runs $\mathsf{ZKPoK}(\vec{\rho}, P^{\vec{\rho}})$ with CA.

CA checks if proofs are correct, then runs $(\sigma, (\vec{C}, \vec{O}), \mathsf{uk}_{k'}) \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{A}, L', \mathsf{pk}_{\mathsf{u}}; (P^{\rho_j})_{j\in[2]})$, where $(\vec{C}, \vec{O}) \leftarrow \mathsf{CSCA.Commit}_2(A_i, \alpha, P^{\rho_j})_{i\in[2]}$ and $\mathsf{nym}_{\mathsf{u}} = \mathsf{pk}_{\mathsf{u}}$. Sets $\mathsf{dk}_{L'} = \mathsf{uk}_{k'}$ and outputs $(\sigma, \mathsf{dk}_{L'})$ as well as $(\vec{C}, \vec{O})$.

U checks if $\Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{u}}, \vec{C}, \sigma, (\vec{T}, \vec{U})) = 1 \wedge \Sigma.\mathsf{UKVerify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{L'}, L', \sigma) = 1$, where $(\vec{T}, \vec{U}) = (\mathsf{A}, \vec{O})$, sets $\vec{O} = \vec{\rho}$ and runs $(\sigma', (\vec{C}', \vec{O}'), \mathsf{dk}'_{L'}, \mathsf{nym}'_{\mathsf{u}}, \chi) \leftarrow \Sigma.\mathsf{ChangeRep}(\mathsf{nym}_{\mathsf{u}}, \mathsf{dk}_{L'}, (\vec{C}, \vec{O}), \sigma, \mu, \psi)$ for $\mu, \psi \leftarrow \mathbb{Z}_p^*$, update $\mathsf{aux}_{\mathsf{u}}$ with χ, ψ and saves a credential $\mathsf{cred}_{\mathsf{u}} = \sigma'$, as well as $(\mathsf{dk}'_{L'}, \vec{C}', \vec{O}')$.

Issuing/delegating a credential (an issuer U **and a receiver** R**)** $[\mathsf{IssueCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{L'}, \mathsf{sk}_{\mathsf{u}}, \mathsf{cred}_{\mathsf{u}}, A_I, L'') \leftrightarrow \mathsf{ReceiveCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{r}}, A_I)] \rightarrow (\mathsf{cred}_{\mathsf{r}}, \mathsf{dk}'_{L''})$:

On input $(\mathsf{nym}_{\mathsf{r}}, A_I, \mathsf{cred}_{\mathsf{u}} = \sigma, \mathsf{dk}_{L'}, (\mathsf{A}, \vec{C}, \vec{O}))$, U prepares a delegated credential for $\mathsf{nym}_{\mathsf{r}}$ and $\mathsf{A}' = (\mathsf{A}, A_I)$:

- Run $\sigma' \leftarrow \Sigma.\mathsf{ConvertSig}(\mathsf{vk}, \mathsf{sk}_{\mathsf{u}}, \mathsf{sk}_{\mathsf{u}'}, \sigma)$ with R, run $(\sigma'', (\vec{C}', O_I), \mathsf{uk}_{k''}) \leftarrow \Sigma.\mathsf{ChangeRel}(M_I, \sigma', \vec{C}, \mathsf{uk}_{k'}, L'')$, where L'' is a level, $\mathsf{uk}_{k'} = \mathsf{dk}_{L'}$, $\mathsf{uk}_{k''} = \mathsf{dk}_{L''}$ and $M_I = A_I$ for $\vec{C}' = (\vec{C}, C_I)$.
- Send $(\sigma'', \vec{C}', \vec{O}' = (\vec{O}, O_I))$ and optionally $\mathsf{dk}_{L''} = \mathsf{uk}_{k''}$ to R.

R checks $\Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{nym}_{\mathsf{r}}, \vec{C}', \sigma'', (\mathsf{A}', \vec{O}')) = 1$, then for $\mu, \psi \leftarrow \mathbb{Z}_p^*$ runs $(\sigma''', (\vec{C}'', \vec{O}''), \mathsf{dk}'_{L''}, \mathsf{pk}'_{\mathsf{u}}, \chi) \leftarrow \Sigma.\mathsf{ChangeRep}(\mathsf{nym}_{\mathsf{r}}, \mathsf{dk}_{L''}, (\vec{C}', \vec{O}'), \sigma'', \mu, \psi)$ and updates $\mathsf{aux}_{\mathsf{r}}$ with χ, ψ and stores $(\mathsf{cred}_{\mathsf{r}} = \sigma''', (\vec{C}'', \vec{O}''))$ and the delegatable key $\mathsf{dk}'_{L''}$.

Showing of a credential (a prover P **and a verifier** V**)** $\mathsf{CredProve}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{nym}_{\mathsf{P}}, \mathsf{aux}_{\mathsf{P}}, \mathsf{cred}_{\mathsf{P}}, D) \leftrightarrow \mathsf{CredVerify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{nym}_{\mathsf{P}}, D) \rightarrow (0, 1)$:

On input a credential $\sigma = \mathsf{cred}_{\mathsf{r}}$ for $\mathsf{nym}_{\mathsf{r}}$ and $D = \{d_j\}_{j\in[k]}$, P prepares a proof for $\mathsf{nym}_{\mathsf{P}}$ and $\{A_j\}_{j\in[k]}$:

- Run $(\sigma', (\vec{C}', \vec{O}'), \mathsf{pk}'_{\mathsf{u}}, \chi) \leftarrow \Sigma.\mathsf{ChangeRep}(\mathsf{pk}_{\mathsf{u}}, \perp, (\vec{C}, \vec{O}), \sigma, \mu, \psi)$ for $\mu, \psi \leftarrow \mathbb{Z}_p^*$, where $\sigma = \mathsf{cred}_{\mathsf{r}}$ and $\mathsf{pk}_{\mathsf{u}} = \mathsf{nym}_{\mathsf{r}}$. Set $\sigma' = \mathsf{cred}_{\mathsf{P}}$ and $\mathsf{pk}'_{\mathsf{u}} = \mathsf{nym}_{\mathsf{P}}$, and update $\mathsf{aux}_{\mathsf{P}}$ with ψ, χ.
- Run $W_j \leftarrow \mathsf{CSCA.OpenSubset}(C_j, A_j, O_j, d_j)$ for $j \in [k]$. Aggregate witness $\pi \leftarrow \mathsf{CSCA.AggregateAcross}(\{C_j, d_j, W_j\}_{j\in[k]})$.
- Send $(\mathsf{cred}_{\mathsf{P}}, \vec{C}', \mathsf{nym}_{\mathsf{P}}, \pi)$ to V, and run $\mathsf{ZKPoK}((\mathsf{sk}_{\mathsf{P}}, \mathsf{aux}_{\mathsf{P}}), \mathsf{nym}_{\mathsf{P}})$ with V.

V outputs 1, if $\mathsf{ZKPoK}((\mathsf{sk}_{\mathsf{P}}, \mathsf{aux}_{\mathsf{P}}), \mathsf{nym}_{\mathsf{P}})$ verifies and $\Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{u}}, \vec{C}, \sigma, (T, \vec{U})) = 1$, where $\sigma = \mathsf{cred}_{\mathsf{P}}$, $T = D$, $\vec{U} = \pi$ and $\mathsf{pk}_{\mathsf{u}} = \mathsf{nym}_{\mathsf{P}}$. Else output 0.

Figure 2: Our DAC **scheme (**Σ **denotes our** SPSEQ-UC **scheme from Section 3.3).**

On compactness of credentials. Credentials are constant-size as the 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. However, we should consider the commitment randomness as auxiliary data. Each commitment randomness represents a scalar that allows recomputing the commitment. So it is sufficient to store the randomness vector that is the length of $O(L)$ for a fixed (and typically very small) L.

On (non-)interactive zero-knowledge proofs. In Fig 2, we use a NIZK proof in Setup and interactive ones elsewhere. To ensure the unforgeability of DAC, we need to extract from every sequential query to O^{RootIss}. Thus as in [23] we rely on black-box extractable interactive ZKPoK to avoid potential blow-ups in rewinding. Straight-line extractable NIZK [20, 32] are an alternative option at the cost of additional computation. While technically not required, we also use interactive ZKPoK for showing a credential.

Sharing of credentials. As inherent in (D)AC systems, we cannot prevent users from sharing their credentials with others. In our case this also covers switching them to other keys (as this is a core functionality in our approach). It is possible to prevent the switching of keys by ordinary users in the delegation chain by removing X_0 from the vk (thus $\mathsf{pk}_{\mathsf{CA}}$) and only handing x_0 to delegators as a special delegation key. Preventing this type of sharing is however possible by using secure elements for secret key operations (cf. [28]). An interesting open question is whether one can extend the user public keys to define equivalence classes (as done in [17]) in a way that one can always compute a canonical representative of the respective secret key class. This secret could be tied to a valuable secret outside the system to disincentivize sharing (cf. [8]).

Restricting power of delegation. For a vector of set commitments and openings $(\vec{C}, \vec{O})$ corresponding to a credential, a delegator can decide to withhold elements from $\vec{O}$. This prevents the delegatee from being able to open the corresponding commitments and thus showing the respective attributes.

5 IMPLEMENTATION AND EVALUATION

We now present an evaluation of our our SPSEQ-UC and DAC schemes implement as a Python library [10]. Our implementation is based upon the `bplib`library[11] and petlib[12] with `OpenSSL` bindings[13]. It uses the BN256 curve, providing efficient type 3 pairings at around 100 bit security. We also want to point to an independent Rust implementation of our scheme[14].

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 is based upon Schnorr-style discrete-logarithm zero-knowledge proofs and using Damgard's technique [16] for obtaining malicious-verifier interactive zero-knowledge proofs of knowledge during the showing and issuing/delegating of credentials. It also uses NIZK proofs 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

[10] https://github.com/mir-omid/DAC-from-EQS
[11] https://github.com/gdanezis/bplib
[12] https://github.com/gdanezis/petlib
[13] https://github.com/dfaranha/OpenPairing
[14] https://github.com/docknetwork/crypto/tree/main/delegatable_credentials.

parameters: ℓ represents an upper bound for the length of commitment vector, t an upper bound for the cardinality of committed sets, $n_i < t$ is 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 (each commitment) for simplicity. Moreover, k represents the length of attribute set vector $\mathsf{A} = (A_1, \ldots, A_k)$ and thus commitment vector $\vec{C}$. The number of attribute sets which can be delegated is k'. The results are shown

Table 2: Running times for SPSEQ-UC and DAC (ms)

	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

in Table 2, where μ_{AV} is the mean and SD the standard deviation of 100 executions of each algorithm. Note that in each delegation one additional attribute set is added. More precisely, in Table 2, we set the above parameters as $t = 25$, $\ell = 15$, $k = 4$, $k' = 7$ and $n = 10$ to cover a broad spectrum of applications. ChangeRep/uk represents the randomization of a 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 and D all d_i's, i.e., $D = (d_i)_{i \in [k]}$. We thereby assume that each subset d_i contains approximately half of the attributes in A_i. Let us provide an example 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 in DAC) 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 $|D| = 4$. By that, we say the total number of attributes is $|\vec{M}| = |\mathsf{A}| = k \cdot n$. We use this semantics in Fig 3 and 4.

In Fig 3 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 $|\vec{M}|$ 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.

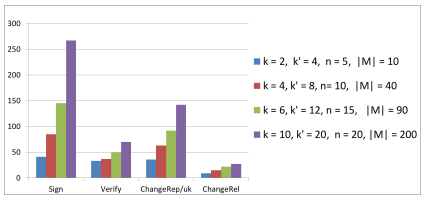
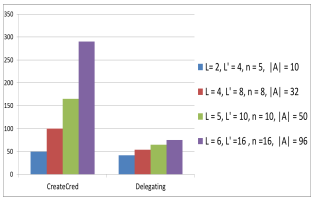
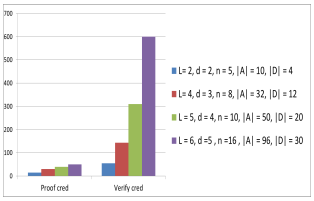


Figure 3: The running times of SPSEQ-UC (ms)

In Fig 4 we show the effect of increasing the parameters (k, k', d, n) on the computation time of DAC such that $k = L$ and $k' = L'$ here mean for levels. We measure the computation time of CredProve and CredVerify protocols for n, L and d (a disclosed attribute subset) varying from 5 to 16, 2 to 6 and 2 to 5, respectively. The total disclosed attributes length $|D| = d \cdot L$ and the total attributes length $|\mathsf{A}| = n \cdot L$ range from 4 to 30 and 10 to 96, respectively. The CredProve and CredVerify are independent of L'. Meanwhile, the computation time of CreateCred and IssueCred change when L' varies from 4 to 16 in addition to being dependent on n and L. These algorithms are independent of d. As can be seen in Fig 4, the pairing



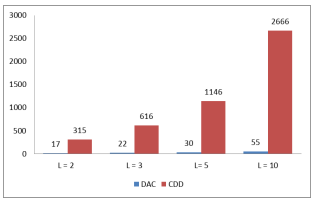
(a) The running times of IssueCred



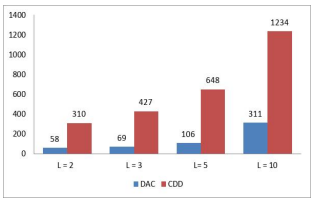
(b) The running times of CredProve

Figure 4: The running times of DAC (ms)

product 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 also suitable for time-critical applications like public transportation or ticketing.



(a) The running times of CredProve (ms)



(b) The running times of CredVerify (ms)

Figure 5: Comparison between our DAC and CDD (n = 4)

Comparison with CDD [5, 7]. In Fig. 5, we present an approximate comparison of running time between our DAC and CDD, which has recently been implemented[15] by [5]. Their implementation is in Go and, similar to ours, uses the BN256 curve. In [5] running times for proving and verifying credentials from benchmarks on a c2-standard-60 GCE VM running Ubuntu 18.04 (60 vCPUs, Intel Cascade Lake 3.1 GHz, 240 GB RAM) are provided. To make the approaches comparable, we set parameters as in [5] as follows: $n = 4$ attributes per level (the maximum in [5]) and L for levels from 2 to 10. We note that the machine used to obtain the benchmarks in [5] is much more powerful than the one used to benchmark our DAC. But we still significantly outperform CDD. We note that benchmarking them on the same platform will only increase the computational advantage of our approach.

[15] https://github.com/IBM/dac-lib

ACKNOWLEDGMENTS

We are very grateful to the anonymous reviewers for their many helpful comments and suggestions. We also want to thank Scott Griffy and Lovesh Harchandani for helpful feedback on the paper and pointing out a number of typos. 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. We 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), the Horizon Europe research programme under grant agreement n°101073821 (Sunrise), and by the Austrian Science Fund (FWF) and netidee SCIENCE under grant agreement P31621-N38 (Profet). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] Balthazar Bauer and Georg Fuchsbauer. 2020. Efficient Signatures on Randomizable Ciphertexts. In *SCN 20 (LNCS, Vol. 12238)*, Clemente Galdi and Vladimir Kolesnikov (Eds.). Springer, Heidelberg, 359–381. https://doi.org/10.1007/978-3-030-57990-6_18

[2] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. 2009. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO 2009 (LNCS, Vol. 5677)*, Shai Halevi (Ed.). Springer, Heidelberg, 108–125. https://doi.org/10.1007/978-3-642-03356-8_7

[3] Johannes Blömer and Jan Bobolz. 2018. Delegatable Attribute-Based Anonymous Credentials from Dynamically Malleable Signatures. In *ACNS 18 (LNCS, Vol. 10892)*, Bart Preneel and Frederik Vercauteren (Eds.). Springer, Heidelberg, 221–239. https://doi.org/10.1007/978-3-319-93387-0_12

[4] Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. 2021. Issuer-Hiding Attribute-Based Credentials. In *CANS 2021 (LNCS, Vol. 13099)*, Mauro Conti, Marc Stevens, and Stephan Krenn (Eds.). Springer, 158–178.

[5] Dmytro Bogatov, Angelo De Caro, Kaoutar Elkhiyaoui, and Björn Tackmann. 2021. Anonymous Transactions with Revocation and Auditing in Hyperledger Fabric. In *Cryptology and Network Security - 20th International Conference, CANS*, Vol. 13099. Springer, 435–459. https://doi.org/10.1007/978-3-030-92548-2_23

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

[7] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. 2017. Practical UC-Secure Delegatable Credentials with Attributes and Their Application to Blockchain. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 683–699. https://doi.org/10.1145/3133956.3134025

[8] Jan Camenisch and Anna Lysyanskaya. 2001. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In *EUROCRYPT 2001 (LNCS, Vol. 2045)*, Birgit Pfitzmann (Ed.). Springer, Heidelberg, 93–118. https://doi.org/10.1007/3-540-44987-6_7

[9] Jan Camenisch and Anna Lysyanskaya. 2003. A Signature Scheme with Efficient Protocols. In *SCN 02 (LNCS, Vol. 2576)*, Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano (Eds.). Springer, Heidelberg, 268–289. https://doi.org/10.1007/3-540-36413-7_20

[10] Jan Camenisch and Anna Lysyanskaya. 2004. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *CRYPTO 2004 (LNCS, Vol. 3152)*, Matthew Franklin (Ed.). Springer, Heidelberg, 56–72. https://doi.org/10.1007/978-3-540-28628-8_4

[11] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. 2013. Succinct Malleable NIZKs and an Application to Compact Shuffles. In *TCC 2013 (LNCS, Vol. 7785)*, Amit Sahai (Ed.). Springer, Heidelberg, 100–119. https://doi.org/10.1007/978-3-642-36594-2_6

[12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. 2014. Malleable Signatures: New Definitions and Delegatable Anonymous Credentials. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*. IEEE Computer Society, 199–213. https://doi.org/10.1109/CSF.2014.22

[13] Melissa Chase and Anna Lysyanskaya. 2006. On Signatures of Knowledge. In *CRYPTO 2006 (LNCS, Vol. 4117)*, Cynthia Dwork (Ed.). Springer, Heidelberg, 78–96. https://doi.org/10.1007/11818175_5

[14] Melissa Chase, Trevor Perrin, and Greg Zaverucha. 2020. The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 1445–1459. https://doi.org/10.1145/3372297.3417887

[15] Aisling Connolly, Pascal Lafourcade, and Octavio Perez-Kempner. 2022. Improved Constructions of Anonymous Credentials from Structure-Preserving Signatures on Equivalence Classes. In *PKC 2022 (LNCS, Vol. 13177)*, Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe (Eds.). Springer, 409–438.

[16] Ronald Cramer, Ivan Damgård, and Philip D. MacKenzie. 2000. Efficient Zero-Knowledge Proofs of Knowledge Without Intractability Assumptions. In *PKC 2000 (LNCS, Vol. 1751)*, Hideki Imai and Yuliang Zheng (Eds.). Springer, Heidelberg, 354–372. https://doi.org/10.1007/978-3-540-46588-1_24

[17] Elizabeth C. Crites and Anna Lysyanskaya. 2019. Delegatable Anonymous Credentials from Mercurial Signatures. In *CT-RSA 2019 (LNCS, Vol. 11405)*, Mitsuru Matsui (Ed.). Springer, Heidelberg, 535–555. https://doi.org/10.1007/978-3-030-12612-4_27

[18] Elizabeth C. Crites and Anna Lysyanskaya. 2021. Mercurial Signatures for Variable-Length Messages. *PoPETs* 2021, 4 (Oct. 2021), 441–463. https://doi.org/10.2478/popets-2021-0079

[19] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *PoPETs* 2018, 3 (2018), 164–180. https://doi.org/10.1515/popets-2018-0026

[20] Marc Fischlin. 2005. Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors. In *CRYPTO 2005 (LNCS, Vol. 3621)*, Victor Shoup (Ed.). Springer, Heidelberg, 152–168. https://doi.org/10.1007/11535218_10

[21] Georg Fuchsbauer. 2011. Commuting Signatures and Verifiable Encryption. In *EUROCRYPT 2011 (LNCS, Vol. 6632)*, Kenneth G. Paterson (Ed.). Springer, Heidelberg, 224–245. https://doi.org/10.1007/978-3-642-20465-4_14

[22] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. 2015. Practical Round-Optimal Blind Signatures in the Standard Model. In *CRYPTO 2015, Part II (LNCS, Vol. 9216)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.). Springer, Heidelberg, 233–253. https://doi.org/10.1007/978-3-662-48000-7_12

[23] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. 2019. Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials. *Journal of Cryptology* 32, 2 (April 2019), 498–546. https://doi.org/10.1007/s00145-018-9281-4

[24] Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. 2020. Pointproofs: Aggregating Proofs for Multiple Vector Commitments. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 2007–2023. https://doi.org/10.1145/3372297.3417244

[25] Jens Groth. 2015. Efficient Fully Structure-Preserving Signatures for Large Messages. In *ASIACRYPT 2015, Part I (LNCS, Vol. 9452)*, Tetsu Iwata and Jung Hee Cheon (Eds.). Springer, Heidelberg, 239–259. https://doi.org/10.1007/978-3-662-48797-6_11

[26] Jens Groth and Amit Sahai. 2008. Efficient Non-interactive Proof Systems for Bilinear Groups. In *EUROCRYPT 2008 (LNCS, Vol. 4965)*, Nigel P. Smart (Ed.). Springer, Heidelberg, 415–432. https://doi.org/10.1007/978-3-540-78967-3_24

[27] Christian Hanser and Daniel Slamanig. 2014. Structure-Preserving Signatures on Equivalence Classes and Their Application to Anonymous Credentials. In *ASIACRYPT 2014, Part I (LNCS, Vol. 8873)*, Palash Sarkar and Tetsu Iwata (Eds.). Springer, Heidelberg, 491–511. https://doi.org/10.1007/978-3-662-45611-8_26

[28] Lucjan Hanzlik and Daniel Slamanig. 2021. With a Little Help from My Friends: Constructing Practical Anonymous Credentials. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 2004–2023. https://doi.org/10.1145/3460120.3484582

[29] ISO/IEC 18013-5. 2021. Personal identification – ISO-compliant driving licence – Part 5: Mobile driving licence (mDL) application. International Standard.

[30] Malika Izabachène, Benoît Libert, and Damien Vergnaud. 2011. Block-Wise P-Signatures and Non-interactive Anonymous Credentials with Efficient Attributes. In *13th IMA International Conference on Cryptography and Coding (LNCS, Vol. 7089)*, Liqun Chen (Ed.). Springer, Heidelberg, 431–450.

[31] Mojtaba Khalili, Daniel Slamanig, and Mohammad Dakhilalian. 2019. Structure-Preserving Signatures on Equivalence Classes from Standard Assumptions. In *ASIACRYPT 2019, Part III (LNCS, Vol. 11923)*, Steven D. Galbraith and Shiho Moriai (Eds.). Springer, Heidelberg, 63–93. https://doi.org/10.1007/978-3-030-34618-8_3

[32] Yashvanth Kondi and abhi shelat. 2022. Improved Straight-Line Extraction in the Random Oracle Model With Applications to Signature Aggregation. Cryptology ePrint Archive, Report 2022/393. https://ia.cr/2022/393.
[33] Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. 2020. Anonymous Tokens with Private Metadata Bit. In *CRYPTO 2020, Part I (LNCS, Vol. 12170)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, 308–336. https://doi.org/10.1007/978-3-030-56784-2_11
[34] Sinisa Matetic, Moritz Schneider, Andrew Miller, Ari Juels, and Srdjan Capkun. 2018. DelegaTEE: Brokered Delegation Using Trusted Execution Environments. In *27th USENIX Security*. 1387–1403.
[35] David Pointcheval and Olivier Sanders. 2016. Short Randomizable Signatures. In *CT-RSA 2016 (LNCS, Vol. 9610)*, Kazue Sako (Ed.). Springer, Heidelberg, 111–126. https://doi.org/10.1007/978-3-319-29485-8_7
[36] Olivier Sanders. 2020. Efficient Redactable Signature and Application to Anonymous Credentials. In *PKC 2020, Part II (LNCS, Vol. 12111)*, Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas (Eds.). Springer, Heidelberg, 628–656. https://doi.org/10.1007/978-3-030-45388-6_22
[37] Victor Shoup. 1997. Lower Bounds for Discrete Logarithms and Related Problems. In *EUROCRYPT'97 (LNCS, Vol. 1233)*, Walter Fumy (Ed.). Springer, Heidelberg, 256–266. https://doi.org/10.1007/3-540-69053-0_18

A ADDITIONAL PRELIMINARIES

A.1 Set Commitments

Definition A.1 (Set commitment [23]). A set commitment scheme SC consists of the following PPT algorithms.

- $\mathsf{SC.Setup}(1^\lambda, 1^t) \to \mathsf{pp}_{\mathsf{sc}}$: 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 $\mathsf{pp}_{\mathsf{sc}}$ (which include a description of an efficiently samplable message space S_{SC} containing sets of maximum cardinality t). $\mathsf{pp}_{\mathsf{sc}}$ 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 $S \in S_{\mathsf{SC}}$. 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 $S \in S_{\mathsf{SC}}$, it outputs 1, and 0 otherwise.
- $\mathsf{SC.OpenSubset}(C, S, O, T) \to W$: Takes as input a commitment C, a set $S \in S_{\mathsf{SC}}$, 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 [23] 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 [16] and made non-interactive using different techniques (discussed below).

ZKPoK. Let $L_{\mathcal{R}} = \{x \mid \exists\, w : (x, w) \in \mathcal{R}\} \subseteq \{0,1\}^*$ be a formal language, where $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ 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 A.2 (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 A.3 (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 $(\mathcal{P}, \mathcal{V}^*)(x)$ shows the transcript of the communication between $\mathcal{P}$ and $\mathcal{V}^\star$ on the common input x.

Definition A.4 (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, when having sequential executions one can use interactive ZKPoK. It must also be noted that when one is willing to pay some extra costs, one could instead use straight-line extractable NIZK, e.g., obtained via Fischlin's transformation [20, 32] or use the "encryption to the sky paradigm" (with a public key derived via a random oracle).

B CSCA CONSTRUCTION

Our Cross Set commitment aggregation scheme CSCA is defined as follows:

- $\mathsf{CSCA.Setup}(1^\lambda, 1^t) \to \mathsf{pp}_{\mathsf{CSCA}}$: 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 $H : \{0,1\}^* \to \mathbb{Z}_p$, pick $\alpha \leftarrow \mathbb{Z}_p$, 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 $S_{\mathsf{CSCA}} = \{S \subset \mathbb{Z}_p | 0 < |S| \leq t\}$. $\mathsf{pp}_{\mathsf{CSCA}}$ will be an implicit input to all algorithms.
- $\mathsf{CSCA.Commit}(S_j) \to (C_j, O_j)$: On input a set $S_j \in S_{\mathsf{SC}}$: pick $\rho_j \leftarrow \mathbb{Z}_p$, compute $C_j \leftarrow (P^{fs_j(\alpha)})^{\rho_j} \in \mathbb{G}_1^*$ 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 $S_j \in S_{\mathsf{SC}}$, α, and P^{ρ_j}: compute $C_j \leftarrow (P^{\rho_j})^{fs_j(\alpha)} \in \mathbb{G}_1^*$ and output (C_j, O_j) with $O_j \leftarrow \perp$.

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 $\rho_j \notin \mathbb{Z}_p^*$ or $S_j \notin S_{\mathsf{CSCA}}$ then return $\perp$. Otherwise if $O_j = \rho_j$ and $C_j = (P^{f_{S_j}(\alpha)})^{\rho_j}$, return 1; else return 0

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 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 $W_j \leftarrow (P^{f_{S_j \setminus T_j}(\alpha)})^{\rho_j}$.

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 $T_j \notin S_{\mathsf{SC}}$, 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.

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:

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

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 $Z_S(\alpha) = \prod_{i\in S}(\alpha - i)$.

Randomization. We can randomize π and commitments C_j as $(\pi^\mu, \vec{C}^\mu)$ 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 [6], 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 [24]), meaning that their analysis carries over.

C THEORETICAL COMPARISON TO CDD

In this section, we compare the asymptotic efficiency of our DAC with CDD [7].

C.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 [7] in Table 3. We use notations that are used in [7], 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 $\vec{M}$ and commitment $\vec{C}$ 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 $\mathsf{uk}_{k'} = \mathsf{dk}_{L'}$. We summarize the efficiency analysis as follows, where we also count the cost of the $\vec{C}$ 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:

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

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 [7], we assume that each commitment is the output of one delegation-level, e.g., if k = 2 such that $\vec{C} = (C_1, C_2)$, the $L = 2$ is delegation level. In other words, at each delegation level, we add one attribute set and the related commitment. Also, unlike [7] 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 [7].

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 $|\mathbb{G}_1| = |\mathbb{Z}_p| = 256$ 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 $\mathsf{uk}_{k'} = \mathsf{dk}_{L'}$ in our scheme is $z \cdot 256$.

Table 3: Computational complexity

	CDD [7]	Ours
CredProve	**odd:** $\sum_{i=1,3}^{L} 1\mathbb{G}_2 + (n_i+2)\mathbb{G}_1 + (1+d_i)\mathbb{G}_t^2$ $+(1+u_i)\mathbb{G}_t^3 + (2+n_i)\mathbb{G}_1^2$	$((k+3)\mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_1^2) +$
	even: $\sum_{i=2,4}^{L} 1\mathbb{G}_1 + (n_i+2)\mathbb{G}_2 + (1+d_i)\mathbb{G}_t^2$ $+(1+u_i)\mathbb{G}_t^3 + (2+n_i)\mathbb{G}_2^2$	$\left(\mathbb{G}_1^{\lvert D\rvert}\right) + \left(\sum_{i=1}^{\lvert D\rvert}(\mathbb{G}_1^{u_i} + \mathbb{G}_1)\right)$
CredVerify	$(1+d_1)E + (3+u_1+d_L)E^2 + u_L E^3$ $+(4+n_1+d_L)\{\mathbb{G}_t\}+$	$\left(E^k + E^2 + 4E\right) +$
	$\sum_{i=1}^{L}((1+d_i)E^2 + (1+u_i)E^3 + (1+d_i)\{\mathbb{G}_t\})$	$\left(E + E^k + \mathbb{G}_2^{\lvert S\rvert} + \sum_{i=1}^{\lvert D\rvert}\left(\mathbb{G}_2^{\lvert S-d_i\rvert} + \mathbb{G}_2\right)\right)$

C.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: $((k+5)|\mathbb{G}_1| + |\mathbb{G}_2| + \mathbb{Z}_p)$, where k is the size of $\vec{C}$ (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
Ours	$((k+5)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert + \mathbb{Z}_p)$
CDD [7]	$(2L + \sum_{i=1,3}^{L-1}(n_i+u_i))\lvert\mathbb{G}_1\rvert +$ $(2L-1+\sum_{i=2,4}^{L}(n_i+u_i))\lvert\mathbb{G}_2\rvert + 2\mathbb{Z}_p$

D ADDITIONAL DEFINITIONS

D.1 Correctness of SPSEQ-UC

Subsequently, we state all the single correctness requirements.

Definition D.1 (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 $k \le k' \le \ell$, 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 $\vec{M}$, $\vec{\rho}$, $\vec{T} \subseteq \vec{M}$, all $(\sigma, (\vec{C}, \vec{O}), \mathsf{uk}_{k'}) \in \mathsf{Sign}(\mathsf{sk}, \vec{M}, k', \mathsf{pk}_u; \vec{\rho})$, any $\vec{U}$ with $1 = \mathsf{SC}.\mathsf{VerifySubset}(C_j, T_j, U_j)_{j\in k}$ (for U_j being a subset opening) and $1 = \mathsf{SC}.\mathsf{Open}(C_i, T_i, U_i)_{j\in k}$ (for U_j being an opening):

Verification: We have that:

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

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

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

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

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

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

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

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

D.2 Class-Hiding

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

Definition D.2 (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:

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

D.3 DAC Oracle Description and Security Properties

We use $\langle O \rangle$ to denote the collection of oracles in the games. For the anonymity game we have $\langle O \rangle = (O^{\mathsf{User}}, O^{\mathsf{Corrupt}}, O^{\mathsf{CreateRoot}}, O^{\mathsf{ObtIss}}, O^{\mathsf{Obtain}}, O^{\mathsf{RootObt}}, O^{\mathsf{CredProve}})$ and for the unforgeability game $\langle O \rangle = (O^{\mathsf{User}}, O^{\mathsf{CreateRoot}}, O^{\mathsf{Corrupt}}, O^{\mathsf{ObtIss}}, O^{\mathsf{RootIss}}, O^{\mathsf{Issue}}, O^{\mathsf{CredProve}})$.

Moreover, 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. Moreover, in each issuing query ($O^{\mathsf{ObtIss}}, O^{\mathsf{Issue}}$) only one commitment (and an attribute set) is added in the commitment vector. Also, for simplicity, we assume that cred_i contains $(\sigma, (\vec{C}, \vec{O}, \mathsf{pk}_i = \mathsf{nym}_i), \mathsf{aux}_i)$.

$O^{\mathsf{User}}(i)$: 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 $\mathsf{pk}_i = \mathsf{nym}_i$.

$O^{\mathsf{Corrupt}}(i, \mathsf{pk}_i)$: 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 $(i, \mathsf{A}, \mathsf{dk}_{L'}, \mathsf{cred}_i)$ of $\mathcal{L}_{\mathsf{cred}}[i]$. Finally, it sets the form $(\perp, \mathsf{A}, L', \perp) \in \mathcal{L}_{\mathsf{cred}}$ for all A and i in this case.

$O^{\mathsf{CreateRoot}}(i, L', A)$: Takes as input a user identity i, a level L' and attributes A. If $i \notin \mathcal{HU}$ it returns $\perp$, else it creates a root credential by running

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

with a user i for an attribute set A and appends $(i, A, \mathsf{dk}_{L'}, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$.

$O^{\mathsf{RootIss}}(L', \mathsf{A})$: Takes as input an Level L' and attributes A. It creates a root credential by running the CreateCred protocol with $\mathcal{A}$: $\mathsf{CreateCred}(L', \mathsf{A}, \mathsf{sk}_{\mathsf{CA}}) \leftrightarrow \mathcal{A}$ for an attribute set A and appends $(\perp, \mathsf{A}, L', \perp)$ 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.

$O^{\mathsf{RootObt}}(i, L', A)$: On input a user identity i, a level L' 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 $(i, \mathsf{A}, \mathsf{dk}_{L'}, \mathsf{cred}_i) \in \mathcal{L}_{\mathsf{cred}}$.

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

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

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

$O^{\mathsf{Obtain}}(j, A_l, L'')$: On input a user identity $j \in \mathcal{HU}$ and a set of attributes A_l, and (optionally) a level L''. 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 $(\mathsf{cred}_j, \mathsf{dk}'_{L''}, \mathsf{A}')$ and it appends $(j, \mathsf{A}', \mathsf{dk}'_{L''}, \mathsf{cred}_j)$ 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.

$O^{\mathsf{Issue}}(i, A_l, L'')$: On input a user identity i, a set of attributes A_l, and (optionally) a level L''. If $i \notin \mathcal{HU}$ it returns $\perp$. Also it checks if $\nexists(i, \mathsf{A}, \mathsf{dk}_{L'}, \mathsf{cred}_i) \in \mathcal{L}_{\mathsf{cred}}$, returns $\perp$. Else, it runs: $\mathsf{IssueCred}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{dk}_{L'}, \mathsf{sk}_i, \mathsf{cred}_i, A_l, L'') \leftrightarrow \mathcal{A}$. The elements $(\perp, \mathsf{A}' = (\mathsf{A}, A_l), L'', \perp)$ 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.

$O^{\mathsf{CredProve}}(j, D)$: On input an index of an issuance j and subsets D. This oracle first parses $\mathcal{L}_{\mathsf{cred}}[j]$ as $(i, \mathsf{A}', \mathsf{dk}'_{L''}, \mathsf{cred}_i)$. Let cred_i be the credential issued on A' for a user i during the i-th query to O^{ObtIss} or O^{Obtain} (or it can be outputs of $O^{\mathsf{CreateRoot}}$ 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.

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, $\mathsf{cred}_b \leftarrow O^{\mathsf{ObtIss}}$ 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 [17], 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 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. The essence of the game is captured by the oracles O_b^{Anon} in Fig 6. 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 D.3 (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 6, 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 6) if he is able to convince an honest verifier that he satisfies a certain policy while does not have an appropriate credential.

Definition D.4 (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 6.

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

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

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

- 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 $D(\mathsf{A}'_0) \neq D(\mathsf{A}'_1) \vee |\vec{C}_0| \neq |\vec{C}_1| \vee \mathsf{cred}_b \nleftarrow O^{\mathsf{ObtIss}}$, **return** $\perp$.
- Otherwise run: $\mathcal{A} \leftrightarrow \mathsf{CredProve}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{sk}_b, \mathsf{nym}_b, \mathsf{aux}_b, \mathsf{cred}_b, D)$

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

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

Figure 6: Experiments $\mathsf{ExpAno}_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t)$ and $\mathsf{ExpUnf}_{\mathsf{DAC},\mathcal{A}}(\lambda,\ell,t)$.

E PROOFS

E.1 Security of DAC

Lemma E.1 (Unforgeability). *Let* ZKPoK *be a ZKPoK and let* SPSEQ-UC *be unforgeable, then the* DAC *construction in Fig 2 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 E.3) with non-negligible probability and let $(\vec{C}^*, \mathsf{nym}^*_P, \mathsf{A}^*, \sigma^*)$ be the message-signature pair it uses and $\vec{W}^*$ be the witness for an attribute set $(D^*, \mathsf{A}') \notin \mathcal{R}_{L'}$ (this implies $D^* \notin \mathsf{A}'$), for all $i = \perp$ where $i, \mathsf{A}', \mathsf{dk}_{L'} \in \mathcal{L}_{\mathsf{cred}}$; 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 C in the unforgeability game for SPSEQ-UC and $\mathcal{B}$ simulates the DAC-unforgeability game for $\mathcal{A}$. 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 $\mathsf{sk}_{\mathsf{CA}}$:

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

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 $(\vec{C}^*, \sigma^* = (Z^*, Y^*, \hat{Y}^*, T^*), D^*, \mathsf{nym}^*_P, \vec{W}^*)$ 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 $(D^*, \mathsf{A}') \notin \mathcal{R}_{L'}$. In all cases, this means that $(\mathsf{sk}^*_P, (\vec{C}^*, D^*, \vec{W}^*), \sigma^*)$ is a valid forgery against our signature scheme, $\mathcal{B}$ breaks thus unforgeability of SPSEQ-UC which concludes our proof. □

Lemma E.2 (Anonymity). *Let* ZKPoK *be a ZKPoK,* NIZK *be knowledge sound, the* DDH *assumption holds and the* SPSEQ-UC *provides Origin-hiding, Conversion-privacy and Derivation-privacy, then the* DAC *construction in Fig 2 is anonymous.*

Proof. The proof follows a sequence of games until a game where answers for the query to O^{Anon}_b 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 ChangeRep, ChangeRel and 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 O^{Anon}_b. Finally, in $\mathbf{Game}_5$ we replace the respective commitment vectors with random vectors.

$\mathbf{Game}_0$: The original game as given in Definition D.3.

$\mathbf{Game}_1$: 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.

$\mathsf{Game}_0 \rightarrow \mathsf{Game}_1$: The success probability in Game_1 is the

same as in Game_0, unless the extractor fails, i.e., using knowledge soundness we have

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

Game_2: As Game_1, except that the experiment runs O_b^{Anon} as follows: Like in Game_1, but for $\mu, \psi \in \mathbb{Z}_p^*$, all executions of $\text{ChangeRep}(\text{pk}_u, \text{dk}_{L'}, (\vec{C}, \vec{O}), \sigma, \mu, \psi)$ for the credential $(i_b, \text{dk}'_{L''}, \text{A}', \sigma_b) \leftarrow \mathcal{L}_{\text{cred}}[\text{j}_b]$ are replaced by $\text{Sign}(\text{sk}_{\text{CA}}, \text{A}', k', \text{pk}_u; \vec{\rho})$. So, oracles are simulated as in Game_1 , except for the following oracles as:

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

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

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

Game_3: As Game_2, except that the experiment runs O_b^{Anon} as follows: All proofs $\text{ZKPoK}(\text{sk}_p, \text{nym}_p)$ and $\text{ZKPoK}(\vec{\rho}, P^{\vec{\rho}})$ in CredProve and CreateCred respectively, are simulated.

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

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

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 O^{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 O^{User}) and runs $O^{\text{User}}, O^{\text{Corrupt}}$ and O_b^{Anon} as follows:

- $O^{\text{User}}(i)$: As in Game_3, except if this is the ω-th call to the oracle then it additionally defines $i^* \leftarrow i$.
- $O^{\text{Corrupt}}(i, \text{pk}_u)$: If $i \in \mathcal{CU}$ or $i \in O_b^{\text{Anon}}$, 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.
- $O_b^{\text{Anon}}(\text{j}_0, \text{j}_1, D)$: As in Game_3, except that if $i^* \neq i_b \leftarrow \mathcal{L}_{\text{cred}}[\text{j}_b]$, the experiment stops and outputting $b' \leftarrow \{0, 1\}$.

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

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

Game_5: As Game_4, except that for $O_b^{\text{Anon}}(\text{j}_0, \text{j}_1, D)$: it picks $\vec{C} \leftarrow (\mathbb{G}_1^*)^k$ and performs the showing using $\text{cred}' \leftarrow (\vec{C}, \text{Sign}(\text{sk}, \vec{M}, \ldots))$, 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 $\vec{C}$; while $\vec{W}$ is distributed as in Game_4, in particular, they are unique elements satisfying $\text{VerifySubset}(C_i, D_i, W_i)_{i \in [k]}$.

$\text{Game}_4 \rightarrow \text{Game}_5$: Let $(\text{BG}, P^x, P^y, P^z)$ be a DDH instance (not to be confused with SPSEQ-UC elements) for $\text{BG} = \text{BGGen}(1^\lambda)$ where $x, y \leftarrow \mathbb{Z}_p$ 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:

- $O^{\text{ObtIss}}(i, \text{A})$: 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 $\alpha \notin \text{A}$, it computes $C_i \leftarrow f_{A_i}(a) \cdot P^{x_i}$ for $A_i \in \text{A}$ (all C_i are thus distributed as in the original game.)
- $O^{\text{CredProve}}(j, D)$: As in Game_4, with the difference that if $i^* = i \leftarrow \mathcal{L}_{\text{cred}}[j]$, 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 $D = (d_i)_{i \in [k]}$.)
- $O_b^{\text{Anon}}(\text{j}_0, \text{j}_1, D)$: As in Game_4, with the following difference. Using self-reducibility of DDH, it picks $s, t \leftarrow \mathbb{Z}_p^*$ 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 O_b^{Anon} it occurs that any $Y'_i = 1_{\mathbb{G}_1}$ or $Z'_i = 1_{\mathbb{G}_1}$ 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 O_b^{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 $\epsilon_{\text{DDH}}(\lambda)$ denote the advantage of solving the DDH problem and q_l the number of queries to the O_b^{Anon}, we have

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

In Game_5 the O_b^{Anon} oracle returns a fresh signature σ on random elements $\vec{C} \leftarrow (\mathbb{G}_1^*)^k$ 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

$$\Pr[S_4] \leq \Pr[S_5] + \epsilon_{\text{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p} = \frac{1}{2} + \epsilon_{\text{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p},$$

$$\Pr[S_3] \leq \frac{1}{2} + q_u \cdot \Pr[S_4] - \frac{1}{2} \cdot q_u \leq \frac{1}{2} + q_u \cdot (\epsilon_{\text{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p}),$$

$$\Pr[S_0] \leq \Pr[S_1] + ks(\lambda) \leq \frac{1}{2} + ks(\lambda) + q_u \cdot (\epsilon_{\text{DDH}}(\lambda) + (1 + 2q_l)\frac{1}{p})$$

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

E.2 Privacy Notions of SPSEQ-UC

We now prove that our SPSEQ-UC construction from Section 3.3 is Origin-hiding (Def. 3.4) and provides Conversion-privacy (Def. 3.5) and Derivation-privacy (Def. 3.6).

Lemma E.3 (Origin-hiding). *The construction described in Section 3.3 is Origin-hiding.*

Proof. Origin-hiding of ChangeRep follows from the perfect adaptation of SPSEQ [23]. 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 $\vec{C} \in (\mathbb{G}_1^*)^k, \vec{T} \subseteq \vec{M}$, any $\vec{O}, \vec{U}$ 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 $y \in \mathbb{Z}_p^*$, a signature $(Z, Y, \hat{Y}, T) \in \mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{G}_1$ satisfying $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \vec{C}, (Z, Y, \hat{Y}, T), (\vec{T}, \vec{U})) = 1$ 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 $\mu, \psi \in \mathbb{Z}_p^*$, $\mathsf{ChangeRep}(\mathsf{pk}_u, \mathsf{uk}_{k'}, (\vec{C}, \vec{O}), (Z, Y, \hat{Y}, T), \mu, \psi)$ 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 $\mathsf{Verify}(\mathsf{vk}, (\mathsf{pk}_u \cdot P^\chi)^\psi, \vec{C}^\mu, \sigma', (\vec{T}, \vec{U})) = 1$. 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 $\mu, \psi \in \mathbb{Z}_p^*$, 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 $\chi \leftarrow \mathbb{Z}_p^*$ 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:

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

Lemma E.4 (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 $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{KeyGen}(\mathsf{pp})$, $(\mathsf{sk}_u, \mathsf{pk}_u) \in \mathsf{UKeyGen}$, $\vec{C}, \vec{T}, \vec{M}, \vec{U}, \vec{O}$ s.t. $\mathsf{SC.Open}(C_j, M_j, O_j)_{j\in k} = 1$ and σ. If $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \vec{C}, \sigma, (\vec{T}, \vec{U})) = 1$. 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 E.5 (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, $\vec{M}$, $\vec{O} = \vec{\rho}$, k', k'', $\vec{T}$, $\vec{U}$, $\mathsf{uk}_{k'}$, and σ. If $\mathsf{SC.Open}(C_j, M_j, O_j)_{j\in k} = 1 \wedge \mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \vec{C}, \sigma, (\vec{T}, \vec{U})) = 1 \wedge \mathsf{UKVerify}(\mathsf{vk}, \mathsf{uk}_{k'}, k', \sigma) = 1$, then for an index $l = k + 1 \in [k + 1, k']$, let M_l be a message set such that the message vector is $\vec{M}^* = (\vec{M}, M_l)$ and the related commitment vector is $\vec{C}^* = (\vec{C}, C_l)$. We intend to show that ChangeRel produces outputs with the same distribution as Sign for vectors $\vec{M}^*$ and $\vec{C}^*$: $\mathsf{Sign}(\mathsf{sk}, \vec{M}^*, k', \mathsf{pk}_u; \vec{\rho}) \approx \mathsf{ChangeRel}(M_l, \sigma, \vec{C}, \mathsf{uk}_{k'}, k'')$. More precisely, for some $y \in \mathbb{Z}_p^*$, 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 $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_u, \vec{C}^*, \sigma, (\vec{T}, \vec{U})) = 1$ 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:

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

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

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 $\vec{M}^*$ using the randomness y. This is, for vectors $\vec{C}^*, \vec{M}^*$, ConvertSig produces signatures with the same distribution as Sign. □

E.3 Unforgeability of SPSEQ-UC

Our main technical result is to prove that our scheme satisfies unforgeability (Def. 3.3) in the generic group model (GGM) [37] 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 E.1. *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 7 that only uses generic group operations on the group elements it receives. After getting the public parameter $(\alpha^o P, \alpha^o \hat{P})_{0\le o\le t}$, 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 $\left(\left(U_{j,o}^{(i)}\right)\right)_{j\in[k^{(i)}+1,k'^{(i)}],o\in[t]}$ 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

$$C_h^* = \kappa^{(h)}P + \sum_{j=1}^{q}\left(\kappa_{z,j}^{(h)}Z_j + \kappa_{s,j}^{(h)}S_j + \kappa_{t,j}^{(h)}T_j + \sum_{o=0}^{t}\sum_{m=k^{(j)}+1}^{k'^{(j)}}\kappa_{m,j,o}^{(h)}U_{m,o}^{(j)}\right) + \sum_{o=1}^{t}\kappa_{a,o}^{(h,q+1)}a^oP + \kappa_{x,0}^{(h)}X_0 + \sum_{u=1}^{b}\kappa_{\mathsf{pk},u}^{(h)}P_a$$

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

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

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

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

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

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

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

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:

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

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

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

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

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

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

Figure 7: Experiment $\mathsf{ExpUnfStat}_{\mathsf{SPSEQ\text{-}UC},\mathcal{A}}(\lambda,\ell,t)$.

$$\begin{aligned}
&\sigma + \sum_{j=1}^{q}\left(\sigma_{z,j}z_j + \sigma_{s,j}s_j + \sigma_{t,j}t_j + \sum_{o=0}^{t}\sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}}\sigma_{m,j,o}u_{m,o}^{(q+1)}\right)\\
&+ \sum_{o=1}^{t}\sigma_{a,o}a^o + \sigma_{x,0}x_0 + \sum_{u=1}^{b}\sigma_{\mathsf{pk},u}p_a\\
=&\phi + \sum_{j=0}^{\ell}\phi_{x,j}x_j + \sum_{j=1}^{q}\phi_{s,j}s_j + \sum_{o=1}^{t}\phi_{a,o}a^o \qquad (6)\\
&\tau + \sum_{j=1}^{q}\left(\tau_{z,j}z_j + \tau_{s,j}s_j + \tau_{t,j}t_j + \sum_{o=0}^{t}\sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}}\tau_{m,j,o}u_{m,o}^{(j)}\right)\\
&+ \sum_{o=1}^{t}\tau_{a,o}a^o + \tau_{x,0}x_0 + \sum_{u=1}^{b}\tau_{\mathsf{pk},u}p_a\\
=&x_1\left(\sigma + \sum_{j=1}^{q}\left(\sigma_{z,j}z_j + \sigma_{s,j}s_j + \sigma_{t,j}t_j + \sum_{o=0}^{t}\sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}}\sigma_{m,j,o}u_{m,o}^{(q+1)}\right)\right.\\
&\left.+ \sum_{o=1}^{t}\sigma_{a,o}a^o + \sigma_{x,0}x_0 + \sum_{u=1}^{b}\sigma_{\mathsf{pk},u}p_a\right)\\
&+ x_0\left(\psi^{(q+1)} + \sum_{j=1}^{q}\left(\psi_{z,j}^{(q+1)}z_j + \psi_{s,j}^{(q+1)}s_j + \psi_{t,j}^{(q+1)}t_j\right.\right.\\
&\left.+ \sum_{o=0}^{t}\sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}}\psi_{m,j,o}^{(q+1)}u_{m,o}^{(j)}\right)\\
&\left.+ \sum_{o=1}^{t}\psi_{a,o}^{(q+1)}a^o + \psi_{x,0}^{(q+1)}x_0 + \sum_{u=1}^{b}\psi_{\mathsf{pk},u}^{(q+1)}p_a\right) \qquad (7)
\end{aligned}$$

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:[16]

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

From (2) and (3), over this quotient we have and $t_i = z_i = u_{m,o}^{(j)} = 0$ 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:

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

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

$$\begin{aligned}
&\sum_{j=1}^{q}\left(\sigma_{z,j}z_j + \sigma_{t,j}s_jx_1 + \right.\\
&\left.\sum_{o=0}^{t}\sum_{m=k^{(q+1)}+1}^{k'^{(q+1)}}\sigma_{m,j,o}u_{m,o}^{(q+1)}\right) \mod (x_0) = \sum_{j=1}^{\ell}\phi_{x,j}x_j \qquad (13)
\end{aligned}$$

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:

$$\forall j > 0 : \phi_{x,j} = 0. \qquad (14)$$

Then the equation (13) becomes:

[16] 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.

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

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 $(x_1,\ldots,x_\ell)$:

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

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:

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

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

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

Then by noticing for all j, we know $\mathsf{pk}^{(j)}$ 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 $k>j$:

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

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

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

Then (7) modulo (x_0) becomes:

$$\sum_{j=1}^{q}\left(\tau_{z,j}z_j+\tau_{t,j}x_1s_j\right)\mod(x_0)=x_1\left(\sigma+\sum_{j=1}^{q}\left(\sigma_{s,j}s_j\right)+\sum_{o=1}^{t}\sigma_{a,o}a^o\right) \tag{26}$$

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, $\left(\sum_{j=1}^{i_0} \phi_{s,j} s_j\right)$ 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:

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

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):

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

It becomes:

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

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 $\sigma_{s,i_0} \neq 0$

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

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

It implies that $\left(\vec{M}_{i_0}, \vec{T}^*\right) \notin \mathcal{R}_{k^{(i_0)}}$, thus $\exists j_0 \in \{1, \ldots, k^{(i_0)}\}$, such that $T_{j_0} \notin M_{j_0}^{(i_0)}$. 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:

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

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 $\frac{c\left(o+q(4+\ell\cdot t)\right)^2(2q+t+1)}{p.}$