Practical Delegatable Attribute-Based Anonymous Credentials with Chainable Revocation

Min Xie[1], Peichen Ju[1], Yanqi Zhao[3], Zoe L. Jiang[1,2], Junbin Fang[4], Yong Yu[5], and Xuan Wang[1,2]

[1] School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Shenzhen 518055, China.
[2] Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, Guangdong, China.
[3] School of Cyberspace Security, Xi'an University of Posts and Telecommunications, Xi'an, 710121, China.
[4] School of Science and Technology, Jinan University, Guangzhou 510632, China.
[5] School of Computer Science, Shaanxi Normal University, Xi'an, 710062, China.

Abstract. Delegatable Anonymous Credentials (DAC) are an enhanced Anonymous Credentials (AC) system that allows credential owners to use credentials anonymously, as well as anonymously delegate them to other users. In this work, we introduce a new concept called Delegatable Attribute-based Anonymous Credentials with Chainable Revocation (DAAC-CR), which extends the functionality of DAC by allowing 1) fine-grained attribute delegation, 2) issuers to restrict the delegation capabilities of the delegated users at a fine-grained level, including the depth of delegation and the sets of delegable attributes, and 3) chainable revocation, meaning if a credential within the delegation chain is revoked, all subsequent credentials derived from it are also invalid.
We provide a practical DAAC-CR instance based on a novel primitive that we identify as structure-preserving signatures on equivalence classes on vector commitments (SPSEQ-VC). This primitive may be of independent interest, and we detail an efficient construction. Compared to traditional DAC systems that rely on non-interactive zero-knowledge (NIZK) proofs, the credential size in our DAAC-CR instance is constant, independent of the length of delegation chain and the number of attributes. We formally prove the security of our scheme in the generic group model and demonstrate its practicality through performance benchmarks.

Keywords: Structure-preserving signatures · Delegatable anonymous credentials · Attribute-based credentials · Chainable revocation · Fine-grained delegation.

1 Introduction

Privacy-preserving attribute-based credentials (PABCs) [1], originally introduced as anonymous credentials [2, 3], enable users to authenticate themselves

to service providers in a privacy-preserving manner, revealing only the necessary information for completing transactions. Increasing legal requirements for better protection of personal data, as well as growing security concerns, have led to the emergence of various anonymous credential schemes [4–6, 18, 24].

While anonymous credentials provide strong privacy features, they do not fully protect user privacy in practice. They assumes that the verifier (e.g., an access provider) knows the public keys of the credential issuers. This leakage reveals details about the user, such as their location, organization, or business unit. Moreover, in practice, certificates are usually issued in a hierarchical manner, and in everyday life, we often need to delegate tasks, responsibilities and permissions to others, or we simply want to share our access to resources and services with another person or across our different electronic devices. As highlighted in [8, 14], anonymous credentials do not provide effective functionality when used in hierarchical structures.

Delegatable anonymous credentials. Delegatable anonymous credentials (DAC), introduced by Chase et al. [7] and further improved by Belenkiy et al [8], can solve the above problem. The DAC scheme is rooted in a trusted authority, which issues initial credentials for level $L = 1$. It allows level L users to delegate their credentials to level $L + 1$ users, who can further delegate the same credentials. Similar to anonymous credentials, users can undergo verification without revealing their identity, only by showing the public key of the root issuer without any undisclosed attributes to the verifier. Unfortunately, the use of a large number of zero-knowledge proofs makes the scheme inefficient in practice.

Subsequently, Chase et al. proposed a new delegatable anonymous credential structure [20] following a similar technique as [8]. However, their instantiation of controlled linkable signatures still required Groth-Sahai proofs, resulting in a construction that is essentially as inefficient as Belenkiy et al.'s construction. Camenisch et al. [9] introduced a practical DAC scheme enabling to generate the proof of credential chain ownership in privacy-preserving manner, yet it cannot receive credentials anonymously. Moreover, this structure leads to a linear increase in the credential size with respect to the product of attribute length and delegation depth. Blömer and Bobolz (BB) [19] later proposed another practical DAC method that utilizes dynamically malleable signatures (DMS) and NIZK proofs.However, due to the necessity of hiding DMS signatures and proving the verification relation of DMS (resulting in linear size of undisclosed attribute count), which leads to a complex NIZK statement that is not expected.

DAC from SPSEQ. Crites and Lysyanskaya [10] extended the structure-preserving signature on equivalence classes (SPSEQ) [4, 29] to the equivalence class on the key space to construct a new signature scheme called mercury signature and proposed a possibility is the most efficient and conceptually simplest DAC structure. Unlike previous schemes, SPSEQ does not require additional proof of NIZK. Unfortunately, their scheme does not support attributes in a meaningful way. While Crites and Lysyanskaya improved it in [18] to support attributes, it still does not support selective disclosure, resulting in the disclosure of all attributes during the showing phase. Moreover, the size of credentials

in the scheme grows linearly with the delegation depth. Based on this, Mir and Slamanig et al. [11] proposed a new delegatable attributes-based anonymous credential scheme based on structure-preserving signatures on equivalence classes on updatable commitments (SPSEQ-UC). But it does not support delegations with fine-grained capability constraints on specific set of attributes, and revocation of anonymous credentials.

Current limitations of DAC. Despite the various advances made by the aforementioned works, there are still gaps in terms of more flexible revocation in DAC and the challenge of fulfilling all desired properties simultaneously. On the one hand, the schemes proposed by Chase et al. [7] and Belenkiy et al. [8], although theoretically sound in terms of complexity, are limited in their practicality because they adopt universal zero-knowledge proof, or Groth-Sahai proof, involving expensive pairing operations and large credential sizes. On the other hand, the vast majority of current DAC solutions do not support credential revocation. Although [27–29] supports revocation, they only support the revocation of the lowest-level users or credentials. However, revocation in DAC can occur at any user level except the root issuer. Moreover, existing DAC schemes do not consider delegations with fine-grained capability constraints, and it is unacceptable for the specific set of attributes and the depth to which they can continue to delegate to be out of the control of the delegator.

1.1 Our Contribution

In this paper, we fill the gap in terms of more flexible revocation in DAC and propose an efficient instance that satisfies all the desired properties mentioned above. A comparison of our scheme with previous works is shown in Table 1. In detail, our main contributions are summarized as follows.

Delegatable Attribute-based Anonymous Credentials with Chainable Revocation. We introduce and formalize the concept of Delegatable Attribute-based Anonymous Credentials with Chainable Revocation (DAAC-CR), which extends DAC to support *fine-grained delegation*, *strong anonymity*, and *chainable revocation*. Specifically, a delegator at any level of delegation can delegate anonymous attribute-based credentials to a delegatee, and can restrict the delegation ability to the next level in a fine-grained manner, including the set of delegable attributes and the depth of delegation.

More importantly, a new revocation method is captured, called *chainable revocation*, which supports revocation not only of the lowest-level credentials, but also of any level within the delegation chain. Any level of credential in the chain is revoked, all subsequent credentials derived from that are invalid without any additional privacy risks.

Novel building block. We identify a novel building block, referred to as the Structure-Preserving Signatures on Equivalence Classes of Vector Commitments (SPSEQ-VC), which serves as the main ingredient of DAAC-CR. SPSEQ-VC, similar to usual structure-preserving signatures on equivalence classes [4, 11, 30], but the equivalent message space is the vectors of random vector commitments, supporting the randomization and adapting the signature to the new equivalent

class messages, and provides signature, random commitments, and the opening of subsets of commitment vectors. A key feature of SPSEQ-VC is that the signature from the signer u supports invoking a public key pk'_u of another user u', allowing the signature to be tied to pk'_u, which implies that u can adapted the signature into another valid signature for u'. Moreover, benefiting from a novel design that binds the position mapping between the signature key and the vector message space, we capture fine-grained signature adaptation in vector messages.

Efficient instance. We provide a concrete construction of SPSEQ-VC and instantiate an efficient DAAC-CR scheme based on SPSEQ-VC and accumulators. Our DAAC-CR scheme satisfies all the properties mentioned in (1) and additionally offers the following features: first, it represents a simple and practical construction, does not require extensive zero-knowledge proofs. Second, our scheme is practical in two respects: the size of the credentials is independent of the length of the delegation chain and the number of attributes, and the communication overhead required in showing phase depends only on the depth of the delegation, not on the number of attributes. Moreover, we provide a formal security proof of our scheme in the generic group model. Finally, we implement SPSEQ-VC and DAAC-CR schemes, and perform a comprehensive evaluation showing the practicality.

Table 1: Comparison of practical DAC schemes.

scheme	Attr	SelD	FGD	FGC	CR	REV	SAnon	|Cred|	|Show|
CDD17 [9]	●	●	○	○	○	○	○	$O(nL)$	$O(uL)$
BB18 [19]	●	●	●	○	○	○	○	$O(1)$	$O(u)$
CL19 [10]	○	○	○	○	○	○	○	$O(L)$	$O(L)$
CL21 [18]	●	○	○	○	○	○	○	nL	$O(uL)$
MSB23 [11]	●	●	○	●	○	○	●	$O(1)$	$O(L)$
Ours	●	●	●	●	●	●	●	$O(1)$	$O(L)$

○ Not Support ● Support

Attr: attribute-based credentials. **SelD**: selective disclosure.

FGD: fine-grained delegation on specific attributes.

FGC: fine-grained delegation on the depth of delegation. **CR** : chainable revocation.

REV: limited revocation(only for lowest-level credentials). **SAnon**: strong anonymity.

L: depth of delegation chain. n: # of attributes in the credential. u: # of disclosed attributes.

2 Preliminaries and Notation

Let $\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T$ be cyclic groups of prime order p. A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a map that is efficiently computable. We will use $BG = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow BGGen(1^\lambda)$ to denote a bilinear group generator where p is a prime of bitlength λ. Let ω is a primitive n-th root of unity in $\mathbb{Z}_p^\star$ [16]. Given a finite set S, we denote by $x \leftarrow S$ the action of sampling an element uniformly at random from S.

2.1 Lagrange Polynomial Interpolation

Given n pairs $(x_i, y_i)_{i\in[0,n)}$, we can find or interpolate the unique polynomial $f(X)$ of degree $< n$ such that $f(x_i) = y_i$, $i \in [0, n)$ using Lagrange interpolation [12] in $O(nlog2n)$ time [13] as $f(X) = \sum_{i\in[0,n)} L_i(X)y_i$, where $L_i(X) = \prod_{j\in[0,n)} \frac{X-x_j}{x_i-x_j}$. Recall that a Lagrange polynomial $L_i(X)$ has the property that $L_i(x_i) = 1$ and $L_i(x_j) = 0$, $i, j \in [0, n)$ with $j \neq i$. Note that $L_i(X)$ is defined in terms of the $x_i's$ which, throughout this paper, will be either $(\omega^i)_{i\in[0,n)}$ or $(\omega^i)_{i\in I}, I \subset [0, n)$.

2.2 Accumulator

Cryptographic accumulators [14] represent a finite set X as a single succinct value A_X and for each $x_i \in X$ one can compute a witness π_x, certifying membership of x in X. Universal accumulators additionally support non-membership witnesses $\bar{\pi}_y$ that certify non-membership of a value $y \notin X$. Let X be the accumulator set and $X(s) = \prod_{x\in X}(s + x)$ be the accumulator polynomial.

- $Gen_{ACC}(BG, t) \rightarrow (SK_{ACC}, PK_{ACC})$: Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ be the output of $BG(1^\lambda)$. Assume that s is a trapdoor randomly sampled from $\mathbb{Z}_p$. The public parameters are $pp = (BG, g_1^{s^i}, g_2^{s^i} | 0 \leq i \leq t)$, where t is the maximum capacity of the accumulator. Then, set $SK_{ACC} = s$ and $PK_{ACC} = (BG, g_1^{s^i}, g_2^{s^i} | 0 \leq i \leq t)$.
- $Commit_{ACC}(X) \rightarrow A_X$: The accumulator digest of the set $X = \{x_1, ..., x_{|X|}\}$, is $A_X = g_1^{X(s)}$, where the polynomial $X(s) = \prod_{x\in X}(s + x)$.
- $Add_{ACC}(A_X, X, I, SK_{ACC}, PK_{ACC}) \rightarrow A'_X$: A set of elements I, $I \cap X = \Phi$, can be added to the accumulator and the digest can be updated as: $A'_X = g_1^{\prod_{x_i\in I}(s+x_i)}$.
- $MemProvePos_{ACC}(X, y, SK_{ACC}, PK_{ACC}) \rightarrow \pi_y$: The proof of membership of an element y can be computed as $\pi_y = g_1^{\prod_{x\in X\setminus y}(s+x)}$, where X is the accumulated set.
- $MemVerifyPos_{ACC}(A_X, y, \pi_y, PK_{ACC}) \rightarrow 0/1$: The membership proof of an element $y \in X$ in the accumulator can be verified by performing the following pairing check: $e(\pi_y, g_2^y g_2^s) = e(A_X, g_2)$.
- $NonMemProvePos_{ACC}(X, y, SK_{ACC}, PK_{ACC}) \rightarrow \bar{\pi}_y$: The non-membership proof of $y \cap X = \Phi$ involves computing the Bézout coefficients $\alpha(x)$ and $\beta(s)$ s.t. $\alpha(x) \cdot X(s) + \beta(s) \cdot (s + y) = 1$. As the monomial $(s + y)$ is of degree one, the polynomial $\alpha(s)$ is in fact a constant! Thus, the $\pi_y = (\alpha, g_1^{\beta(s)}) \in (\mathbb{Z}_p, \mathbb{G}_1)$ is the non-membership proof of element y.
- $NonMemVerifyPos_{ACC}(A_X, y, \bar{\pi}_y, PK_{ACC}) \rightarrow 0/1$: The non-membership proof can be verified by checking $e(A_X, g_2^\alpha) \cdot e(g_1^{\beta(s)}, g_2^s g_2^y) = e(g_1, g_2)$.
- $MemProve_{ACC}(X, I, SK_{ACC}, PK_{ACC}) \rightarrow \pi_I$: A batch membership proof for set $I \subseteq X$ is computed as follows: $\pi_I = g_1^{\prod_{x_i\in X\setminus I}(s+x_i)}$.

- $MemVerify_{ACC}(A_X, I, \pi_I, PK_{ACC}) \rightarrow 0/1$: The batch proof can be verified by checking: $e(\pi_I, g_2^{\prod_{x_i \in I}(s+x_i)}) = e(A_X, g_2)$.
- $NonMemProve_{ACC}(X, I, SK_{ACC}, PK_{ACC}) \rightarrow \bar{\pi}_I$: A batch non-membership proof for a set I, where $I \cap X = \Phi$, is $\bar{\pi}_I = (g_2^{\alpha(s)}, g_1^{\beta(s)}) \in (\mathbb{G}_2, \mathbb{G}_1)$ such that $\alpha(s) \cdot X(x) + \beta(s) \cdot \prod_{y_i \in I}(s + y_i) = 1$.
- $NonMemVerify_{ACC}(\bar{\pi_I}, A_X, I, PK_{ACC}) \rightarrow 0/1$: The batch proof can be verified by checking $e(A_X, g_2^{\alpha(s)}) \cdot e(g_1^{\beta(s)}, g_2^{\prod_{y_i \in I}(y_i+s)}) = e(g_1, g_2)$.

2.3 Structure-Preserving Signatures on Equivalence Classes

Structure-preserving signatures (SPS) [17] are pairing-based signatures where the message, signature, and verification key consist of source group elements only (in one or both source groups).The notion of SPS on equivalence classes (SPS-EQ) was introduced by Hanser and Slamanig [4]. The scheme allowed joint randomization of messages and their corresponding signatures.If one considers a prime-order group $\mathbb{G}$ and defines the projective vector space $(\mathbb{G}^\star)^l$, there is a partition into equivalence classes given by the following relation $\mathcal{R} : M \in (\mathbb{G}^\star)^l \sim_R M^\star \in (\mathbb{G}^\star)^l \Leftrightarrow \exists \mu \in Z_p^\star : M^\star = \mu M$. If the discrete logarithm problem is hard in $\mathbb{G}$ and one restricts the vector components to be non-zero, given two vectors M and $M^\star$, it is difficult to distinguish whether they were randomly sampled or if they belong to the same equivalence class.

Definition 1. *An Structure-Preserving Signatures on Equivalence Classes (SPS-EQ) scheme on $(G^\star)^l$ consists of the following PPT algorithms:*

- $BGGen_{\mathcal{R}}(1^\lambda) \rightarrow BG$, a bilinear-group generation algorithm, which on input a security parameter λ outputs an asymmetric bilinear group BG.
- $KeyGen_{\mathcal{R}}(BG, n) \rightarrow (sk, pk)$, on input BG and vector length $n > 1$, outputs a key pair (sk, pk).
- $Sign_{\mathcal{R}}(M, sk) \rightarrow \sigma$, given a representative $M \in (G^\star)^l$ and a secret key sk, outputs a signature σ for the equivalence class $[M]_{\mathcal{R}}$.
- $ChgReq_{\mathcal{R}}(M, \sigma, \mu, pk) \rightarrow (M', \sigma')$, on input a representative $M \in (G^\star)^l$ of class $[M]_{\mathcal{R}}$, a signature σ on M, a scalar μ and a public key pk, returns an updated message-signature pair (M', σ'), where $M' = \mu \cdot M$ is the new representative and σ' its updated signature.
- $Verify_{\mathcal{R}}(M, \sigma, pk) \rightarrow 0/1$, is deterministic and, on input a representative $M \in (G^\star)^n$, a signature σ and a public key pk, outputs 1 if σ is valid for M under pk and 0 otherwise.

2.4 Vector Commitment

Vector Commitment allows one to commit a vector $\mathbf{v} \subseteq \mathbb{Z}_p^n$ in a such way and supports to open it at any position $i \in [n]$. Let $\mathbf{v} = (v_0, ..., v_{n-1})$. The vector commitment, introduced by Tomescu et al. in [16] works as follows:

- $VC.Setup(1^\lambda, n) \rightarrow pp_{VC}$: Run $BG = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow BGGen(1^\lambda)$. Pick $\alpha \leftarrow \mathbb{Z}_p^\star$ and generate n-SDH parameters $(g_1, g_1^\alpha, \cdots, g_1^{\alpha^n}, g_2, g_2^\alpha, \cdots, g_2^{\alpha^n})$. Compute $l_i = g_1^{L_i(\alpha)}$ for $i \in [0, n)$, where $L_i(X) = \prod_{j \in [0,n), j \neq i} \frac{X - \omega^j}{\omega^i - \omega^j}$. Return $pp_{VC} = (BG, (g_1^{\alpha^i})_{i \in [0,n)}, (g_2^{\alpha^i})_{i \in [0,n)}, (l_i)_{i \in [0,n)})$.
- $VC.Commit(pp_{VC}, \mathbf{v}) \rightarrow C$: Compute $C = \prod_{i \in [0,n)} (l_i)^{v_i}$
- $VC.Prove(pp_{VC}, I, \mathbf{v}) \rightarrow \pi_I$: Compute $A_I(X) = \prod_{i \in I}(X - \omega^i)$ and divide $f(X) = \sum_{i \in [0,n)} L_i(X) v_i$ by $A_I(X)$, obtaining a quotient $q(X)$ and a remainder $r(X)$. Return $\pi_I = g_2^{q(\alpha)}$.
- $VC.Verify(pp_{VC}, C, \mathbf{v}_I, I, \pi_I) \rightarrow 0/1$: Compute $A_I(X) = \prod_{i \in I}(X - \omega^i)$ and interpolate $R_I(X)$ such that $R_I(\omega^i) = v_i$ for $i \in I$. Return 1 iff. $e(C, g_2) = e(g_1^{A_I(\alpha)}, \pi_I) \cdot e(g_1^{R_I(\alpha)}, g_2)$.

And a vector commitment satisfies the following properties:

Correctness: An honest prover will always convince an honest verifier. That is, the probability of the following is 1.

$$\Pr\left[VerifyPos(pp_{VC}, C, \mathbf{v}_I, I, \pi_I) \;\middle|\; \begin{array}{r} (pp_{VC}) \leftarrow Setup(1^\lambda, n) \\ (\mathcal{C}) \leftarrow VC.Commit(pp_{VC}, \mathbf{v}) \\ (\pi) \leftarrow VC.ProvePos(pp_{VC}, I, \mathbf{v}) \end{array} \right] = 1$$

Position Binding: No PPT adversary $\mathcal{A}$ can provide two valid openings for a position. That is for every PPT adversary $\mathcal{A}$ the following probability is at most negligible.

$$\Pr\left[\begin{array}{r} VerifyPos(pp_{VC}, C, \mathbf{v}_I, I, \pi_I), \\ VerifyPos(pp_{VC}, C, \mathbf{v}'_I, I, \pi_I) \\ \wedge \mathbf{v}_I \neq \mathbf{v}'_I \end{array} \;\middle|\; \begin{array}{r} (pp_{VC}) \leftarrow Setup(1^\lambda, n) \\ (\mathcal{C}, i, \pi, \pi', \mathbf{v}_I, \mathbf{v}'_I) \leftarrow \mathcal{A}(pp) \end{array} \right] \leq nel$$

The aforementioned commitment as an equivalence class of messages does not support re-randomization. We provide a $VC.Random$ algorithm and slightly modify $VC.Verify$ algorithm to support vector commitment and its openings for re randomization on the equivalent class, which detailed as follows:

- $VC.Random(C, \pi_I, \mu) \rightarrow (C', \pi'_I, r)$: Compute $r \leftarrow H(g_2^\mu)$, where $H : \mathbb{G}_2 \rightarrow \mathbb{Z}_p$ is a hash function. Then, computes $C' \leftarrow C^{\mu + r}$ and $\pi'_I \leftarrow \pi_I^{\mu + r}$.
- $VC.Verify_R(pp_{VC}, C', \mathbf{v}_I, I, \pi'_I, g_2^\mu) \rightarrow 0/1$: Compute $A_I(X) = \prod_{i \in I}(X - \omega^i)$, $r = H(g_2^\mu)$ and interpolate $R_I(X)$ such that $R_I(\omega^i) = v_i$ for $i \in I$. Return 1 iff. $e(C', g_2) = e(g_1^{A_I(\alpha)}, \pi'_I) \cdot e(g_1^{R_I(\alpha)}, g_2^\mu) \cdot e(g_1^{R_I(\alpha)}, g_2^r)$.

Claim 1. *If the vector commitment provided by [16] is binding, position-binding, and hiding, and the hash function H is secure, then the re-randomized commitment C' with the opening π'_I still satisfies binding, position-binding, and hiding.*

Proof. Hiding follows from the fact that μ is random and r is generated by a hash function H, $\mu + r$ is also random. So C' generated by $VC.Random$ and by $VC.Commit(pp_{VC}, \mathbf{v}(\mu + r)) \rightarrow C'$ cannot be distinguished. Because the C' generated by $VC.Commit$ is hidden, the randomized C' generated by $VC.Random$ is also hidden.

Position-Binding. For any specific position I, it is computationally infeasible to open it to two different values simultaneously such that both are accepted as valid by a verifier. Suppose the adversary computes $C^{\star'} = (C^{\star})^{\mu^{\star}+r^{\star}} = VC.commit(\mathbf{v}^{\star})^{\mu^{\star}+r^{\star}}$ and provide an accepting proof $\pi_I^{\star}$ for $(I, \mathbf{v}^{\star})$ where $\mathbf{v}^{\star}[I] \neq \mathbf{v}[I]$. Note that the adversary does not get μ, and so the coeffcient of r can not be gotten. Comparing coefficients on both sides of verification equation, we therefore have

$$\mathbf{v}^{\star}[I](\mu^{\star} + r^{\star}) = \mathbf{v}[I](\mu + r),$$

where $r = H(\mu)$. Now, suppose μ is a random number and H is a collision-resistant hash function, so $(\mu + r)$ is also random. Since $\mathbf{v}^{\star}[I] \neq \mathbf{v}[I]$, the adversary must find the corresponding $\mu^{\star}$ and $r^{\star}$ such that the equation satisfies. However, due to the collision-resistant property of hash functions

$$\Pr[\mathbf{v}^{\star}[I] \neq \mathbf{v}[I] \wedge \mathbf{v}^{\star}[I](\mu^{\star} + r^{\star}) = \mathbf{v}[I](\mu + r)] = 1/p.$$

The adversary cannot generate a valid C' and π_I' by any ways other than $VC.Random(C, \pi_I, \mu)$. Since the original scheme is position-binding, the re-randomization does not introduce a way to alter the specific openings at any position I without detection. The re-randomization scheme still satisfies position binding. From position binding, it is natural to derive that the scheme also satisfies binding.

3 Delegatable Attribute-Based Anonymous Credential System with Chainable Revocation

In this section, we provide the formal definition and security of DAAC-CR. DAAC-CR, similar to [10] [11], except that it adds fine-grained delegation and a new *chainable revocation* functionality. In terms of security, compared to DAC [11], chainable revocation still has anonymity.

3.1 Formal Definitions

Definition 2 (Delegatable Attributes-Based Anonymous Credentials With Chainable Revocation).

A DAAC-CR scheme consists of the following PPT algorithms and interactive protocols:

$Setup(1^{\lambda}, 1^{n}) \rightarrow pp$: On input the security parameters λ and an upper bound n for the cardinality of attributes, outputs public parameters pp for the system.

$CAKeyGen(pp, 1^l) \rightarrow (pk_{CA}, sk_{CA})$: On input pp and the maximum depth of delegation $l > 1$, outputs a public and secret key pair (pk_{CA}, sk_{CA}) for the root issuer(called CA).

$RAKeyGen(pp, 1^t) \rightarrow (pk_{RA}, sk_{RA})$: On input pp and the maximum number of accumulated elements t, outputs the revocation secret key sk_{RA} and public key pk_{RA} for the trusted revocation authority (called RA).

$UKeyGen(pp) \rightarrow (pk_u, sk_u)$: On input pp, output a key pair (pk_u, sk_u), where sk_u is the user's secret key and pk_u is the public key.

$NymGen(pp, pk_u) \rightarrow (nym_u, sk_{nym})$: On input pp and user's public key pk_u, outputs the user's pseudonym num_u and corresponding secret sk_{nym}.

Issuea root credential

$[CreateCred(k', M_1, sk_{CA}, pk_U) \leftrightarrow ReceiveCred(pk_{CA}, sk_U, M_1) \rightarrow (cred, C, uk_{k'}, F_{ID}(X))$: This interactive protocol involves a root issuer CA and a user U. First, CA runs the *CreateCred* function with his security key sk_{CA}, the maximum depth of delegation k' for U, and the user's public key pk_U along with an attributes vector M_1 as inputs. Then U runs *ReceiveCred* with the same attributes vector, CA's public key, and her secret key sk_U as inputs. The protocol outputs a root credential *cred*, a commitment C, a delegatable key $uk_{k'}$, and an identifier polynomial $F_{ID}(X)$. *cred* is a delegatable credential linked to the commitment C, while the latter corresponds to the attributes vector M_1. $F_{ID}(X)$ includes the ID that is bound to the credential *cred*, as well as the delegatable keys $uk_{k'}$ related to the index k' for a user, rooted at sk_{CA}.

Delegate a credential

$[DelegateCred(pk_{CA}, \mathbf{C}, M_L, uk_{k'}, sk_R, cred_R, k'', I_{L-1}, F_{ID}(X)) \leftrightarrow Receiev eCred(pk_{CA}, sk_U, M_L)] \rightarrow (cred_U, \mathbf{C}', uk_{k''}, F'_{ID}(X))$: This interactive protocol involves an issuer R and a receiver U. R executes the *DelegateCred* algorithm, and U manages the *ReceiveCred* side. The common inputs include the public key of CA, and an attributes vector M_L of U. Additionally, the issuer inputs the identifier polynomial $F_{ID}(X)$, his credential $cred_R$, secret key, a delegation key $uk_{k'}$ (optionally), an index $k'' < k'$, and attribute indexes set I_{L-1} that he is authorized to sign. The receiver inputs her own secret key sk_U. Upon completion of the protocol, output the credential $cred_U$ linked to $\mathbf{C}' = (\mathbf{C}, C_L)$, where C_L is a vector commitment for M_L, $(uk_{k''})$ (if further delegation is intended) or $\perp$, and the updated identifier polynomial $F'_{ID}(X)$.

Revoke a credential

$ReceiveToken(F_{ID}(X)) \leftrightarrow CreateToken(F_{ID}(X), A_X, pp) \rightarrow \bar{\pi}$: This interactive protocol involves a revocation authority (RA) running $CreateToken$, and a user who runs *ReceiveToken*. The common input is the identifier polynomial. The RA inputs an accumulator A_X and public parameters pp, and outputs a non-membership proof $\bar{\pi}$.

$Revoke(ID) \rightarrow ACC'$: Run by the RA, this protocol inputs the ID of a credential to be revoked and updates the blacklist accumulator.

Show a credential

$[Show(pp, pk_{CA}, \Phi, sk_U, cred_U, nym_U, M_L, uk_{k'}, F_{ID}(X), M'_L, D) \leftrightarrow Verify(pp, pk_{CA}, \Phi, M'_L, A'_{ID}, \sigma', \mathbf{C}', D)] \rightarrow 0/1$: This protocol involves a prover U show-

ing and a verifier V checking the possession and validity of a credential. The common inputs include the public parameters pp, public key pk_{CA}, the access rules Φ the subvector of attributes M'_L and the set of indexes D in M'_L. Additionally, the prover U inputs his secret key, pseudonym, credential, and all associated information. The verifier V inputs an accumulator A'_{ID}, which is generated by the identifier polynomial $F_{ID}(X)$. The output is either 1 (valid, accepts the proof of possession of the credential, confirms its validity, and M'_L satisfies the access rules Φ) or 0 (invalid).

Definition 3 (Correctness of DAAC-CR). *If the Setup, CAKeyGen, RAKeyGen, UKeyGen and NymGen are executed correctly, and the CreateCred/ReceiveCred and DelegateCred/ReceiveCred protocols are correctly implemented with honestly generated inputs, the system guarantees that the receiver outputs a valid credential cred. And all credentials prior to a valid credential cred in the delegation chain are valid and not revoked, then cred is always accepted by the verifier through the Show/Verify protocol.*

3.2 Security Models

For security, anonymity (ensuring that users cannot be traced when showing, delegating, or receiving credentials) and unforgeability (ensuring that credentials that have not yet been issued, or that are part of a revoked delegation chain, fail to pass verification) are expected.

This system uses oracles to define two security properties: anonymity, unforgeability. We define five 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, $\mathcal{R}_{cred}$ a list of revoked credentials and $\mathcal{L}_{cred}$ a list of user-credential pairs which includes issued credentials and corresponding attributes and to which user they were issued. All of these lists are initialized to the empty set. An adversary has limited access to the oracle model. Also, for simplicity, we assume that $cred_i$ contains $(\sigma, \mathbf{C}, pk_i)$.

$\mathcal{O}^{HU}(i)$: Corresponds to the creation of an honest user with identity identifier i. Takes as input a user identity i. If $i \in \mathcal{HU} \cup \mathcal{CU}$, returns $\perp$, else, creates a fresh honest user with identity i, adds i to $\mathcal{HU}$, and calls $(pk_i, sk_i) \leftarrow UKeyGen(pp)$ to generate the user key pair, then adds it to $\mathcal{L}_{uk}$.

$\mathcal{O}^{CU}(i, pk_i)$: Corresponds to the creation of a corrupted user with identity i. Takes as input a user identity i and a user public key pk_i. If $i \in \mathcal{CU}$, returns $\perp$. If $i \in HU$, it moves the entry corresponding to i from the list of honest users $\mathcal{HU}$ and adds it to the list of corrupted users $\mathcal{CU}$, else, it creates a new corrupted user with identity i and public key pk_i, then adds $i \in \mathcal{CU}$. It returns sk_i and all the associated credentials items $(i, A_i, uk_{k'}, cred_i)$ of $\mathcal{L}_{cred}[i]$.

$\mathcal{O}^{RIO}(i, k', \mathbf{A})$: Takes as input a user identity i, an index k' and attributes $\mathbf{A}$. If $i \in \mathcal{HU}$, returns $\perp$, else it creates a root credential by running

$$[CreateCred(k', \mathbf{A}, sk_{CA}, pk_i) \leftrightarrow ReceieveCred(pk_{CA}, sk_i, \mathbf{A})] \rightarrow (cred_i, uk_{k'})$$

and adds $(i, \mathbf{A}, uk_{k'}, cred_i)$ to $\mathcal{L}_{cred}$.

$\mathcal{O}^{RIS}(k', \mathbf{A})$: Allows an adversary to play a corrupted user to get a root credential from a CA. On input an index k' and attributes $\mathbf{A}$. It creates a root credential by running the $CreateCred$ protocol with $\mathcal{A}$: $CreateCred(k', \mathbf{A}, sk_{CA}) \leftrightarrow \mathcal{A}$ and adds $(i, \mathbf{A}, uk_{k'}, cred_i)$ to $\mathcal{L}_{cred}$.

$\mathcal{O}^{ROB}(i, k', \mathbf{A})$: Allows an adversary to play a corrupted CA to create a root credential to a user with identity i. On input an index k' and attributes A. If $i \notin \mathcal{HU}$, returns $\perp$. Else, it creates a root credential by running the $ReceieveCred$ protocol with $\mathcal{A}$: $\mathcal{A} \leftrightarrow ReceieveCred(pk_{CA}, sk_i, \mathbf{A})$ and adds $(i, \mathbf{A}, uk_{k'}, cred_i)$ to $\mathcal{L}_{cred}$.

$\mathcal{O}^{OIS}(i, j, A_j, k'')$: An honest user with identity i delegate a credential to an honest user with identity j. On input a vector of attributes A_j and an index k''. If $i, j \notin \mathcal{HU}$ or $(i, \mathbf{A}, uk_{k'}, cred_i) \notin \mathcal{L}_{cred}$, it returns $\perp$, else it delegates a credential by running

$$[DelegateCred(sk_i, cred_i, uk_{k'}, k'', A_j, pk_{CA}) \leftrightarrow ReceieveCred(pk_{CA}, sk_j, A_j)] \rightarrow (cred_j, uk_{k''})$$

and adds $(j, \mathbf{A}', uk_{k''}, cred_j)$ to $\mathcal{L}_{cred}$.

$\mathcal{O}^{COB}(i, A_i, k'')$: Allows an adversary to impersonate a malicious issuer and issue credentials to honest users. On input a user identity i, a vector of attributes A_i, and an index k''. If $i \notin \mathcal{HU}$, returns $\perp$, else, the oracle runs the Receive protocol with $\mathcal{A}$: $\mathcal{A} \leftrightarrow ReceiveCred(pk_{CA}, sk_i, A_i)$. If $cred_i = \perp$ the oracle returns $\perp$. Else it stores the resulting output $(cred_i, uk_{k''}, \mathbf{A}')$ and it adds $(i, \mathbf{A}', uk_{k''}, cred_i)$ to $\mathcal{L}_{cred}$.

$\mathcal{O}^{CIS}(i, A_i, k'')$: Allows an adversary to impersonate a malicious user to obtain credentials from the issuer. On input a user identity i, a vector of attributes A_l, and an index k''. If $i \notin \mathcal{HU} \cup (i, A_i, uk_{k'}, cred_i) \notin \mathcal{L}_{cred}$, returns $\perp$, else, the oracle runs the Issue protocol with $\mathcal{A}$:

$$DelegateCred(sk_i, cred_i, uk_{k'}, k'', A_j, pk_{CA}) \leftrightarrow \mathcal{A}$$

Then, it adds $(\perp, \mathbf{A}, uk_{k''}, \perp)$ to $\mathcal{L}_{cred}$.

$\mathcal{O}^{CSH}(i, D)$: An honest user with identity i proves that his credential is valid and satisfy subvector D. If $i \notin \mathcal{HU}$, returns $\perp$. Else, it parses $\mathcal{L}_{cred}[i]$ as $(i, \mathcal{A}', uk_{k'}, cred_i)$. Then it retrieves (pk_i, sk_i) for i from $cred_i$ and $\mathcal{L}_{uk}$ and runs $Show(pk_{CA}, sk_i, nym_i, cred_i, D, A_{ID}, \bar{\pi}) \leftrightarrow \mathcal{A}$, with the adversary playing the role of the verifier.

$\mathcal{O}^{Ano-b}(j_0, j_1, D)$: On input two indexes j_0, j_1 and subvector D. If $j_0 \vee j_1 > |\mathcal{L}_{cred}|$, returns $\perp$. Else, it parses $\mathcal{L}_{cred}[j_b]$ as $(i_b, \mathbf{A}'_i, uk_i, cred_i)$. If $D(\mathbf{A}'_0) \neq D(\mathbf{A}'_1) \vee |\mathbf{C}_0| \neq |\mathbf{C}_1| \vee cred_b \notin \mathcal{O}^{OIS}$, return $\perp$. Otherwise runs $Show(pk_{CA}, sk_b, nym_b, cred_b, D, A_{ID}, \bar{\pi}) \leftrightarrow \mathcal{A}$.

Anonymity. Anonymity requires that a malicious verifier cannot distinguish any two users. The adversary has adaptive access to an oracle that acts as one of the two credential owners in the verification algorithm (depending on bit b) when input two different user indexes i_0 and i_1.

Definition 4 (Anonymity). *For all probabilistic polynomial-time adversaries $\mathcal{A}$, if the advantage $Adv^{Ano} = |Pr[Exp_{\mathcal{A}}^{Ano-0}(1^\lambda) = 1] - Pr[Exp_{\mathcal{A}}^{Ano-1}(1^\lambda) = 1]|$*

is negligible for the experimental $Exp_{\mathcal{A}}^{Ano\text{-}b}$ definition as follows, then we say the scheme satisfies anonymous.

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 or for which he possesses revoked credentials.

Definition 5 (Unforgeability). *For all probabilistic polynomial-time adversaries $\mathcal{A}$, if the advantage $Adv^{Uf} = |Pr[Exp_{\mathcal{A}}^{Uf}(1^\lambda) = 1]|$ is negligible for the experimental $Exp_{\mathcal{A}}^{Uf}$ definition as follows, then we say the scheme satisfies unforgeability.*

$\mathrm{Exp}_{\mathcal{A}}^{\text{Ano-b}}(1^\lambda)$:

$pp \leftarrow Setup(1^\lambda, 1^n)$
$(pk_{CA}, sk_{CA}) \leftarrow CAKeyGen(pp, 1^l)$
$(pk_{RA}, sk_{RA}) \leftarrow RAKeyGen(pp, 1^t)$
$b \xleftarrow{R} \{0,1\}$
$\mathcal{O} := \{\mathcal{O}^{HU}, \mathcal{O}^{CU}, \mathcal{O}^{RIO}, \mathcal{O}^{CIO}, \mathcal{O}^{ROB}, \mathcal{O}^{COB}, \mathcal{O}^{CSH}, \mathcal{O}^{Ano-b}\}$
$(b') \leftarrow \mathcal{A}(pp, pk_{CA}, \mathcal{O})$
return $(b = b')$.

$\mathrm{Exp}_{\mathcal{A}}^{Uf}(1^\lambda)$:

$pp \leftarrow Setup(1^\lambda, 1^n)$
$(pk_{CA}, sk_{CA}) \leftarrow CAKeyGen(pp, 1^l)$
$(pk_{RA}, sk_{RA}) \leftarrow RAKeyGen(pp, 1^t)$
$\mathcal{O} := \{\mathcal{O}^{HU}, \mathcal{O}^{CU}, \mathcal{O}^{RIO}, \mathcal{O}^{CIO}, \mathcal{O}^{RIS}, \mathcal{O}^{CIS}, \mathcal{O}^{CSH}, \mathcal{O}^{REV}\}$
$(cred_i, \pi, \mathbf{A}) \leftarrow \mathcal{A}(pp, pk_{CA}, \mathcal{O})$
$b \leftarrow [\mathcal{A} \leftrightarrow Verify(pk_{CA}, D, A_{ID}, \bar{\pi}, cred_i)]$
if $(i, \perp, \perp, cred_i) \in \mathcal{L}_{cred}$ with $i \in CU$
$\vee\ cred_i \notin \mathcal{R}_{cred} \vee \forall j \in [1, i)\ cred_j \notin \mathcal{R}_{cred}$
$\vee\ A_i$ statisfies with D, **then**
return 0.
else return b.

4 SPSEQ on Vector Commitments

In this section, we propose a novel building block called Structure-Preserving Signatures on Equivalence Classes on Vector Commitments (SPSEQ-VC). This innovation serves as a slight adjustment of SPSEQ-UC [11], manifesting two aspects:

1) The message space is considered as a vector of randomizable vector commitments, enabling the transition of a signature from a vector over vector commitments to its re-randomized signed commitments vector. This redefinition on equivalence classes space over vector commitments implies that actual message vectors are committed to and can be opened in a re-randomized form for specified subsets of messages. This property is eye-catching for many privacy-preserving applications, such as commit-ahead-of-time systems with authenticity, etc. Notely, to enhance scalability of SPSEQ-VC, we expand the message space into two unrelated parts, where an additional message space is designed as part of a function with rerandomization property in the classes of a single projective equivalence relation (which can be ignored if not needed). This is employed in the DAAC-CR instance detailed in Section 5.

2) The SPSEQ-VC extends the signing of class representatives associated with a singular projective equivalence relation $\mathcal{R}^k$ to a family of relations $\mathbb{R}^l$ where $\mathcal{R}^k \in \mathbb{R}^l$ for $1 \leq k \leq \ell$, which is similar to SPSEQ-UC. In SPSEQ-UC, an update key $uk_{k'}$ can be generated by signing a commitment vector of length k with $k \leq k' \leq \ell$ and can be used to adapt a commitment vector C in class $[C]_{\mathcal{R}^k}$ to C' being another class $[C']_{\mathcal{R}^{k'}}$ of equivalence relation. We extend the functionality of update keys $uk_{k',\boldsymbol{I}}$ by binding the position vector $\boldsymbol{I}$ mapping between the update keys with the vector message space in SPSEQ-VC, which fine-grained adaptation on C'.

4.1 Formal Definitions

Definition 6 (SPSEQ-VC scheme). *A SPSEQ-VC scheme for a vector commitment scheme* $\boldsymbol{VC}$ *consists of the following PPT algorithms:*

- $PPGen(1^\lambda, 1^n) \rightarrow (pp)$: On input the security parameter λ and an upper bound n for the cardinality of committed Vectors, output the public parameters pp.
- $KeyGen(pp, 1^l) \rightarrow (vk, sk)$: On input the public parameters pp and a length parameter $l > 1$, output a verification and signing key pair (vk, sk) for root issuer.
- $UKeyGen(pp) \rightarrow (sk_u, pk_u)$: On input the public parameters pp, output a key pair (sk_u, pk_u) for a user u.
- $Random_C(\mathbf{C}, \mathbf{W}, \mu) \rightarrow (\mathbf{C}', \mathbf{W}')$: On input a commitment vector $\mathbf{C} = (C_1, ..., C_k) = ((C_{1,1}, C_{1,2}), ..., (C_{k,1}, C_{k,2}))$ of size $1 \leq k \leq l$ and corresponding opening proofs $\mathbf{W}$. It outputs $\mathbf{C}'$ and $\mathbf{W}'$.
- $Sign(sk, \mathbf{M}, k', pk_u, aux) \rightarrow (\sigma, \mathbf{C}, uk_{k'})$: On input the root signing key sk, an attributes vector $\mathbf{M} = (M_1, \ldots, M_k)$, an index k' where $1 \leq k' \leq l$, a user public key pk_u, and auxiliary data aux (where aux can be arbitrary if necessary), the procedure computes the commitment $\mathbf{C}_1$ for $\mathbf{M}$ and $\mathbf{C}_2$ for aux. If aux is not $\perp$, it then calculates σ for $\mathbf{C} = (\mathbf{C}_1, \mathbf{C}_2)$. Output $(\sigma, \mathbf{C})$ for pk_u and an update key $uk_{k'}$.
- $Random_{PK}(pk_u, \phi, \delta) \rightarrow pk'_u$: On input a user public key pk_u and randomness ϕ, δ, the algorithm outputs the randomized public key pk'_u.
- $Random_{All}(pk_u, uk_{k'}, \mathbf{C}, \sigma, \mu, \phi, \mathbf{W}) \rightarrow (\sigma', \mathbf{C}', uk'_{k'}, pk'_u, \mathbf{W}', \delta)$: On input an user's public key pk_u, a commitment vector $\mathbf{C} = (C_1, \ldots, C_k)$, a signature σ on $\mathbf{C}$, opening proofs $\mathbf{W}$ for $\mathbf{C}$, randomness ϕ and μ, and a delegated key $uk_{k'}$(optionally). It returns an updated signature σ' for $\mathbf{C}'$, corresponding opening proof $\mathbf{W}'$, as well as a randomized user public key pk'_u, If $uk_{k'} \neq \perp$, it also outputs a randomized update key $uk'_{k'}$.
- $SendConvertSig(vk, sk_u, \sigma) \rightarrow (\sigma_{pro})$: The algorithm is run by a user who wants to delegate a signature σ. It takes as input the verification key vk, a secret key sk_u and the signature σ. It outputs a process signature σ_{pro}.
- $ReceiveConvertSig(vk, sk'_u, \sigma_{pro}) \rightarrow \sigma'$: The algorithm is run by a user who receives a process signature. It takes as input the verification key vk, a secret key sk'_u, a process signature σ_{pro}. It outputs a new signature σ' for pk'_u.

- $Delegate(I_{L-1}, M_L, uk_{k'}, \mathbf{C}, \sigma, k'', aux) \rightarrow (\sigma', \mathbf{C}', uk_{k''})$: This algorithm facilitates the delegation of a credential from a level-$(L-1)$ user to a level-L user. It takes input a set of attributes that can be signed by level-$(L-1)$ user, a message vector $M_L = (m_i)_{i \in I}$ for $L = k+1 \in [k']$, a delegate key $uk_{k'}$, a signature σ for a vector of commitments vector $\mathbf{C} = (C_1, ..., C_k)$, an updatable key $uk_{k'}$, an index $k'' \leq k'$ and aux. This algorithm computes a signature σ' for a new commitment vector $\mathbf{C}' = (\mathbf{C}, C_L = (C_{L,1}, C_{L,2}))$, where $C_{L,1}$ is a vector commitment for M_L and $C_{L,1}$ is a commitment for aux. Also, for $k'' \in [L+1, k']$, updates the updatable key for the range $[L+1, k'']$ into $uk_{k''}$. It outputs $(\sigma', \mathbf{C}', uk_{k''})$.
- $Verify(vk, pk_u, \mathbf{C}, \sigma, \mathbf{D}, \mathbf{W}) \rightarrow 0/1$: On input the verification key vk, a user public key pk_u, a commitment vector $\mathbf{C} = (C_1, ..., C_k)$, a signature σ and corresponding openings and proofs pair $(\mathbf{D}, \mathbf{W})$, it checks whether σ is a valid signature for $(\mathbf{C}, pk_u)$ and $\mathbf{D}$ is the correct opening of $\mathbf{C}$.
- $UKVerify(vk, u_{k'}, k', \sigma) \rightarrow 0/1$: On input the vk, $uk_{k'}$, index k' and a the $\sigma = (Z, Y, Y', T)$. Outputs 1 or 0.

4.2 Security Definitions

The correctness of the commitment scheme naturally follows Lagrangian interpolation.Similar to a standard signature scheme, the SPSEQ-VC scheme should also be correct, unforgeable and privacy.

Definition 7 (Correctness). *The scheme for a vector commitment scheme VC and a parameterized family of equivalence relations $\mathbb{R}^l$ for all $l > 1$, is correct if it satisfies the following conditions for all t, λ, k, k' with $k \leq k' \leq l$, for all $pp \in PPGen(1^\lambda, 1^n, 1^l)$, $(vk, sk) \in KeyGen(pp)$, $(pk_u, sk_u) \in UKeyGen(pp)$.*

Verification: For all $\mathbf{M}$, for all $(\sigma, \mathbf{C}, uk_{k'}) \in Sign_{root}(sk, \mathbf{M}, k', pk_u)$, $Verify(vk, pk_u, \mathbf{C}, \sigma, \mathbf{D}, \mathbf{W}) = 1$.

Change of vector commitments representative: For all $(\mu, \phi), (\sigma', uk'_{k'}, \delta) \in Random_{All}(pk_u, uk_{k'}, \mathbf{C}, \sigma, \mu, \phi)$, for all $\mathbf{C}' \in Random_C(\mathbf{C}, \mu)$, for all $pk'_u \in Random_{PK}(pk_u, \phi, \delta)$, $Verify(vk, pk'_u, \mathbf{C}', \sigma', \mathbf{D}, \mathbf{W}') = 1$. Moreover, $\mathbf{C}' \in [\mathbf{C}]_{\mathcal{R}^k}$.

Signature conversion: For all $(pk_u, sk_u) \in UKeyGen(pp)$, $\sigma' \in ConverSig(vk, sk_u, sk_{u'}, \sigma)$, $Verify(vk, pk_{u'}, \mathbf{C}, \sigma', \mathbf{D}, \mathbf{W}) = 1$.

Change of vector commitments: For all $(pk_{u_l}, sk_{u_l}) \in UKeyGen(pp)$, for all $(\sigma', \mathbf{C}', uk_{k''}, F'_{ID}(X)) \in Delegate(M_l, uk_{k'}, \mathbf{C}, \sigma, k'', I, F_{ID}(X)$, with $\mathbf{C}' = (\mathbf{C}, C_l \in VC.commit(M_l))$, $Verify(vk, pk_{u_l}, \mathbf{C}', \sigma', \mathbf{D}', \mathbf{W}') = 1$.

Unforgeability. For unforgeability, we consider the adversary to have access to the signatures of the chosen message set vectors, control over the randomness of the commitment, and the ability to create and compromise user keys. Additionally, the adversary needs to specify the user secret and public keys $(sk^\star, pk^\star)$ for forgery, which we need to make it useful in a DAAC application. Note that since the outputs of $Delegate$ and $Random_{All}$ have the same distribution as $Sign_{root}$, we do not need to provide access to these oracles as the adversary

can accomplish them on their own. To make the proof more general, we let the message space of $Sign_{root}$ be $\mathbf{M} = (M_1)$. To distinguish whether a signature derived using $uk_{k'}$ is obtained from *Delegate* or freshly generated in the *Sign* oracle, we define the following relation $R_{k'}$. Therefore, signatures legitimately generated using *ConvertSig* and $Random_{All}$ are not considered forgeries.

Definition 8. *Let k, l be integers. For any $l \geq k' \geq k$, define the relation $\mathcal{R}_{k'}$ for two vectors $\mathbf{M} = (M_1, ..., M_k)$ and $\mathbf{M}^\star = (M_1^\star, ..., M_{k'}^\star)$ as follows:*

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

The oracle used in the proof is as follows and Q is the set of queries that $\mathcal{A}$ has issued to the signing oracle:

$\mathcal{O}^{Sign}(M, k', pk_u)$: Corresponds to the adversary $\mathcal{A}$ requesting credentials from oracle. On input a vector of attributes M, and an index k' and a user public key pk_u. If $l \leq k' \leq k$, then runs $(\sigma, C, uk_{k'}) \leftarrow Sign_{root}(sk, k', M, pk_u)$ and sets $Q = Q \cup (M, k', pk_u)$, return $(C, \sigma, uk_{k'})$. Else, return $\perp$.

$\mathcal{O}^{HU}(i)$: Corresponds to the creation of an honest user with identity identifier i. Takes as input a user identity i. Run $(pk_i, sk_i) \leftarrow UKeyGen(pp)$ to generate the user key pair, then set $\mathcal{UL} \leftarrow \mathcal{UL} \cup \{(i, pk_i, sk_i)\}$. Return pk_i.

$\mathcal{O}^{CU}(i)$: Corresponds to the creation of a corrupted user with identity i. Takes as input a user identity i. If $i \in \mathcal{UL}$, then delete the item from $\mathcal{UL}$ and return (sk_i, pk_i). Else, returns $\perp$.

Definition 9 (Unforgeability). *For all $(\lambda, n) \in \mathbb{N}$ and $l > 1$, if the advantage $Adv^{Uf}_{\mathcal{A}}{}^{SPSEQ-VC} = |Pr[ExpUnf^{SPSEQ-VC}_{\mathcal{A}}(1^\lambda, 1^l, 1^n, u) = 1]|$ is negligible for all probabilistic polynomial-time adversaries $\mathcal{A}$, then the SPSEQ-VC scheme is unforgeable. The experiment on unforgeability is as follows:*

$\mathrm{ExpUnf}_{\mathcal{A}}^{\text{SPSEQ-VC}}(1^\lambda, 1^l, 1^n, u)$

$Q := \emptyset; \mathcal{UL} := \emptyset, pp \leftarrow PPGen(1^\lambda, 1^l, 1^n, u), (vk, sk) \leftarrow KeyGen(pp)$

$\mathcal{O} := \left\{\mathcal{O}^{Sign}, \mathcal{O}^{HU}, \mathcal{O}^{CU}, \right\}, ((sk_u^\star, pk_u^\star), \mathbf{C}^\star, \sigma^\star) \leftarrow \mathcal{A}^{\mathcal{O}}(vk, pp)$

Return $\forall (pk_u, sk_u) \notin \mathcal{UL}, \forall (\mathbf{M}, k', pk_u) \in Q :$

$(\mathbf{M}, \mathbf{D}^\star) \notin \mathcal{R}_{k'} \wedge (sk_u^\star, pk_u^\star) \in UKeyGen(pp) \wedge Verify(vk, pk_u^\star, \mathbf{C}^\star, \sigma^\star, \mathbf{D}^\star, \mathbf{W}^\star) = 1$

Definition 10 (Origin-hiding). *For all t, λ, k, k' with $k \leq k' \leq l$, all $pp \in PPGen(1^\lambda, 1^n, 1^l)$, $(vk, sk) \in KeyGen(pp)$, $(pk_u, sk_u) \in UKeyGen(pp)$, all M, σ. If $Verify(vk, pk_u, \mathbf{C}, \sigma, \mathbf{D}, \mathbf{W}) = 1$, then for all μ, ϕ, the algorithm $Random_{All}(pk_u, uk_{k'}, \mathbf{C}, \sigma, \mu, \phi)$ output s a uniformly random $\mathbf{C}' \in [\mathbf{C}]_{\mathcal{R}^k}$, uniformly random pk_u' and σ' in the respective spaces.*

Definition 11 (Derivation-privacy). *For all t, λ, k, k' with $k \leq k' \leq l$, all $pp \in PPGen(1^\lambda, 1^n, 1^l)$, $(vk, sk) \in KeyGen(pp)$, $(pk_u, sk_u) \in UKeyGen(pp)$, all M, σ. If $Verify(vk, pk_u, \mathbf{C}, \sigma, \mathbf{D}, \mathbf{W}) = 1$, then for all $k'' \in [k{+}1, k']$, M_L and*

$(\sigma', \mathbf{C}', uk_{k''}, F'_{ID}(X)) \leftarrow ChangeRel(M_L, uk_{k'}, \mathbf{C}, \sigma, k'', I, F_{ID}(X))$, *the following holds:*

$$(vk, sk, pk_u, uk_{k'}, (\sigma', \mathbf{C}', uk_{k''}) \approx (vk, sk, pk_u, uk_{k'}, Sign(sk, \mathbf{M}', k'', pk_u))$$

Definition 12 (Conversion-privacy). *For all* t, λ, k, k' *with* $k \leq k' \leq l$, *all* $pp \in PPGen(1^\lambda, 1^n, 1^l)$, $(vk, sk) \in KeyGen(pp)$, $(pk_u, sk_u) \in UKeyGen(pp)$, *all* M, σ. *If* $Verify(vk, pk_u, \mathbf{C}, \sigma, \mathbf{D}, \mathbf{W}) = 1$, *then for all*$(pk'_u, sk'_u) \in UKeyGen(pp)$, *the following holds:*

$$(vk, sk, pk'_u, (ConvertSig(vk, sk_u, sk'_u, \sigma)) \approx (vk, sk, pk'_u, Sign(sk, \mathbf{M}', k', pk'_u))$$

4.3 Construction

- $PPGen(1^\lambda, 1^n) \rightarrow pp$: Run $BG = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow BGGen(1^\lambda)$ and $pp_{VC} \leftarrow VC.Setup(1^\lambda, n)$, output $pp = \{BG, pp_{VC}\}$.
- $KeyGen(pp, 1^l) \rightarrow (vk, sk)$: For $0 \leq i \leq l$, pick $x \leftarrow (\mathbb{Z}_P^\star), x_i \leftarrow (\mathbb{Z}_P^\star)$ and $x'_i \leftarrow (\mathbb{Z}_P^\star)$, set signing key $sk = ((x, x_0, ..., x_l), (x'_0, ..., x'_l))$, compute the verification key $vk = ((\bar{X}, X, X_0, ..., X_l), (X'_0, ..., X'_l)) = ((g_1^x, g_2^x, g_2^{x_0}, ..., g_2^{x_l}), (g_2^{x'_0}, ..., g_2^{x'_l}))$, output (vk, sk).
- $UKeyGen(pp) \rightarrow (sk_u, pk_u)$: Pick $w_u \leftarrow \mathbb{Z}_P^\star$, set $pk_u \leftarrow g_1^{w_u}$ and $sk_u = w_u$, output (sk_u, pk_u).
- $Sign(sk, \mathbf{M}, k', pk_u, aux) \rightarrow (\sigma, \mathbf{C}, uk_{k'})$: On input signing key sk, a vector of messages $\mathbf{M} = (M_1, ..., M_k)$, an index k' where $1 \leq k' \leq l$, and a user public key pk_u. The procedure is as follows: First, for each $i \in [k]$, run $VC.Commit(sk, pp, M_i) \rightarrow C_{i,1}$ to commit $M_i := (m_{i,j})_{j \in |M_i|}$ as $C_{i,1} = \prod_{j \in |M_i|} (l_j)^{m_{i,j}}$, where l_is are group elements in pp. If aux is not $\perp$, it computes $C_{i,2} = f(aux)$, where $f : \mathbb{Z}_p \rightarrow \mathbb{G}_1$ could be any desired algorithm. Then, σ is computed by selecting a random $y \leftarrow \mathbb{Z}_p^\star$ and calculating:

$$\sigma = (Z \leftarrow (\prod_{i \in [k]} C_{i,1}^{x_i} C_{i,2}^{x'_i})^{\frac{1}{y}}, Y \leftarrow g_1^y, Y' \leftarrow g_2^y, T \leftarrow g_1^{x_0 \cdot y} \cdot pk_u^x)$$

  Furthermore, if $k \neq l$, an update key $uk_{k'}$ for the range between $k+1$ and k' is computed as:

$$uk_{k'} = (usign_1, using_2) = (((l_i^{x_j})^{\frac{1}{y}})_{j \in [k+1, k'], i \in [I]}, (g_1^{x'_j})^{\frac{1}{y}}_{i \in [k+1, k']})$$

  where I denotes the set of attribute indexes that the user can delegate.
  Set $\mathbf{C}_1 = \{C_{i,1}\}_{i \in [k]}$, $\mathbf{C}_2 = \{C_{i,2}\}_{i \in [k]}$, it outputs a signature $(\sigma, \mathbf{C} = (\mathbf{C}_1, \mathbf{C}_2))$, along with an update key $uk_{k'}$, provided $k \neq l$.
- $Random_C(\mathbf{C}, \mathbf{W}, \mu) \rightarrow (\mathbf{C}', \mathbf{W}', r)$: On input $\mathbf{C} = (\mathbf{C}_1, \mathbf{C}_2)$, corresponding opening proofs $\mathbf{W} = (W_1, ..., W_k)$, and randomness μ, output the randomized results $\mathbf{C}' = \mathbf{C}^{\mu+r}, \mathbf{W}' = \mathbf{W}^{\mu+r}$ and r by running $VC.Random(C_{i,1}, W_i, \mu)$, $VC.Random(C_{i,2}, W_i, \mu)$ for all $i \in [k]$.

- $Random_{PK}(pk_u, \phi, \delta) \rightarrow pk'_u$: On input a user public key pk_u and randomness ϕ, δ, outputs the randomized public key $pk'_u = (pk_u \cdot g_1^{\delta})^{\phi}$ with respect to secret key $(\delta + sk_u)\phi$.
- $Random_{All}(pk_u, uk_{k'}, \mathbf{C}, \mathbf{W}, \sigma, \mu, \phi) \rightarrow (\sigma', \mathbf{C}', \mathbf{W}', uk'_{k'}, pk'_u, \delta)$: On input the user public key pk_u, an optionally update key $uk_{k'}$, the commitment vectors $\mathbf{C} = (\mathbf{C}_1, \mathbf{C}_2)$, a signature σ, randomness ϕ and μ and corresponding opening proof $\mathbf{W}$. It begins by sampling $\delta \leftarrow \mathbb{Z}_p^{\star}$, then proceeds to randomize the commitments and public key by executing $(\mathbf{C}', \mathbf{W}', r) \leftarrow Random_C(\mathbf{C}, \mathbf{W}, \mu)$ and $pk'_u \leftarrow Random_{PK}(pk_u, \phi, \delta)$. The signature is updated to:

  $$\sigma' = (Z^{\frac{\mu+r}{\phi}}, Y^{\phi}, Y'^{\phi}, (T \cdot \bar{X}^{\delta})^{\phi})$$

  If $uk_{k'} \neq \perp$, it also computes a randomized update key as follows:

  $$uk'_{k'} = ((usign_{j,1}^{(\mu+r)\cdot\phi^{-1}}, usign_{j,2}^{(\mu+r)\cdot\phi^{-1}})_{j\in[k+1,k']})$$

  Finally, it outputs $(\sigma', \mathbf{C}', \mathbf{W}', (uk'_k or \perp), pk'_u, \delta)$.
- $SendConvertSig(vk, sk_u, \sigma) \rightarrow (\sigma_{pro})$: On input the root issuer public key vk, a secret key sk_u, and the signature σ. It outputs a processed signature:

  $$\sigma_{pro} = (Z, Y, Y', T' = T \cdot (\bar{X}^{sk_u})^{-1})$$

- $ReceiveConvertSig(vk, sk_{u'}, \sigma_{pro}) \rightarrow \sigma'$: On input the root issuer public key vk, a secret key $sk_{u'}$, and a processed signature $\sigma_{pro} = (Z, Y, Y', T')$, it outputs a signature σ' for $pk_{u'}$ as:

  $$\sigma' = (Z, Y, Y', T'' = T' \cdot \bar{X}^{sk_{u'}} = g_1^{x_0 \cdot y} \cdot pk_{u'}^{x})$$

- $Delegate(M_L, uk_{k'}, \mathbf{C}, \sigma, k'', I_{L-1}, aux_L) \rightarrow (\sigma', \mathbf{C}', uk'_{k''})$: On input a message vector M_L for $L = k + 1 \in [k']$, a signature σ for a commitments vector $\mathbf{C}$, an updatable key $uk_{k'}$, an index $k'' \leq k'$, a set of attribute indexes I_{L-1} that the level-$(L-1)$ user can sign, and auxiliary data aux_L, then, it performs the following steps: First, compute a vector commitment V_l using the keys from the L-th component of $uk_{k'}$:

  $$usign_{L,1} = (((l_i)^{x_L})^{\frac{1}{y}}, i \in I_{L-1}), usign_{L,2} = (g_1^{x'_L})^{\frac{1}{y}}$$

  $$V = \prod_{i \in I}((l_i^{x_L \cdot y^{-1}})^{m_i}) \cdot ((g_1^{x'_L})^{\frac{1}{y}})^{aux_L})$$

  Second, update σ for a new commitment vector $\mathbf{C}' = (\mathbf{C}, C_L)$ as:

  $$\sigma' = ((Z \cdot V), Y, Y', T)$$

  Finally, for $k'' \in [L+1, k']$, update the update key $uk_{k''}$ for $[L+1, k'']$:

  $$uk_{k''} = (usign_{j,1} = (((l_i)^{x_j})^{\frac{1}{y}})_{j\in[L+1,k''],i\in[I_L]}, usign_{j,2} = ((g_1^{x'_j})^{\frac{1}{y}})_{j\in[L+1,k'']})$$

  Output $(\sigma', \mathbf{C}', uk_{k''})$.

– $Verify(vk, pk_u, \mathbf{C}, \sigma, \mathbf{D}, \mathbf{W}) \rightarrow 0/1$: On input of the root issuer public key vk, a public key pk_u, a commitment vector $\mathbf{C} = (\mathbf{C}_1, \mathbf{C}_2)$, a signature σ, and corresponding openings and proofs pairs $(\mathbf{D}, \mathbf{W})$ for commitment $\mathbf{C}$, it checks whether σ is a valid signature for $(\mathbf{C}, pk_u)$. It outputs 0 if any of the following checks fail:

$$\prod_{j=1}^{k} e(C_{j,1}, X_j) \cdot e(C_{j,2}, X'_j) = e(Z, Y') \wedge e(Y, g_2) = e(g_1, Y') \wedge e(T, g_2) = e(Y, X_0) \cdot e(pk_u, X)$$

If $(\mathbf{D}, \mathbf{W})) \neq \perp$, then for all $(D_i, W_i) \in (\mathbf{D}, \mathbf{W})$, it runs $VC.Verify(pp, C_i, D_i, W_i)$. It outputs 1 if all checks hold, otherwise, output 0.

– $UKVerify(vk, uk_{k'}, k', \sigma) \rightarrow 0/1$: On input of the root issuer public key vk, an update key $uk_{k'}$, an index k', and a signature $\sigma = (Z, Y, Y', T)$, outputs 1 if the following condition holds; otherwise, it outputs 0:

$$e(\prod_{i \in I} Y_i, \prod_{j \in [k']} X_j) = e(\prod_{j \in [k']} \prod_{i \in I} usign_{j,1,i}, Y') \wedge e(g_1, \prod_{j \in [k']} X'_j) = e(usign_{j,2}, Y')$$

Theorem 1 (Unforgeability). *The construction described above is unforgeable in the generic group model for Type III bilinear groups.*

Theorem 2 (Privacy). *The construction described above is origin-hiding, conversion-privacy and derivation-privacy in the generic group model for Type III bilinear groups.*

The proof of Theorem 1 and Theorem 2 are presented in Appendix B.

5 Delegatable Attribute-Based Anonymous Credential System with Chainable Revocation from SPSEQ-VC

In this section, we present a concrete instantiation of DAAC-CR based on the SPSEQ-VC signature scheme described in Section 4. In addition, we provide a formal security proof to show that all the security properties described in Section 3 are achieved in the generic group model. Moreover, we demonstrate that our instance achieve the desired functionality: *fine-grained delegation*, *chainable revocation* and *compactness*.

5.1 Scheme overview

Our scheme consists of four phases: *initialization*, *issuance*, *delegation* and *show*. Our *initialization* phase is similar to DAC, except that introduces a new entity RA, which maintains a blacklist and publishes its digest. In *issuance* phase, a Level-1 user sends an anonymous request to CA and obtains a credential from CA, who may also provide a delegation key for further delegation, along with

the fine-grained manner for the set of delegable attributes and the depth of delegation. In *delegation* phase, with the delegation key and credentials, Level-1 users can anonymously issue credentials to Level-2 users by the delegation key. Note that the delegate key comes from CA, which implies that credentials only support constrained attributes and depths specified by CA. Similarly, if a Level-2 user is granted delegation ability and is required to hold the updated delegation key from Level-1 user, it indicates that the Level-1 user can restrict the Level-2 user in the same way. Clearly, there is a fine-grained and flexible delegation capacity transfer, until depth exceeds the limit or delegation capacity is not granted.

In *show* phase, Credential hold by a Level-L user, aimed at satisfying specific access criteria, involves a two-step process: (1) demonstrating the non-revocation of all credentials within the delegation chain; and (2) selectively revealing attributes that satisfy the access criteria and proving that these disclosed attributes are issued in his credential.

To support the above, we use SPSEQ-VC signature over attributes to support fine-grained delegations. An interesting extension designed in SPSEQ-VC is another additional message space, which is used to bind a unique identifier of a credential in our DAAC-CR. Then, we can easily capture chain revocation through a universal accumulator as follows: (1) RA can revoke the credentials by simply adding a unique identifier to the blacklist accumulator. (2) User generates a whitelist accumulator that includes the identifier of his credential and the identifiers of credential derived from them in delegation chain. (3) User proves that all elements in the whitelist accumulatort do not appear in the blacklist. Clearly, we eliminate the need for explicit zero-knowledge proofs for non-membership proofs, which can be verified in batches in the accumulator.

5.2 Concrete construction

Initialize. This phase is run by the root issuer (CA), the trusted revocation authority (RA), and users. We use **SPSEQ-VC** (denoted by Σ) detailed in Section 4 as the signature scheme, **ACC** $= (Gen_{ACC}, Commit_{ACC}, Add_{ACC}, MemProve_{ACC}, MemVerify_{ACC}, NonMem\ Prove_{ACC}, NonMemVerify_{ACC})$ as the universal universal accumulator scheme [14] and $ZKPok = (ZKPok.Setup, ZKPok.Prove, ZkPok.Verify)$. Detailed steps for initialization are as follows.

Issue a Root Credential:(by a Root Issuer CA and a User U). Corresponds the
CreateCred/*ReceiveCred* interactive protocol. The details are as follows.

- U generates a pseudonym $(nym_U, sk_{nym}) \leftarrow NymGen(pp, pk_U)$ and a proof $\pi_{nym} \leftarrow ZKPok.Prove(sk_U, nym_U)$, then sends (nym_U, π_{nym}) along with an attributes vector A_1 to CA.
- CA sets function f in the $Sign$ algorithm as $f(\text{aux}) = g_1^{s+\text{aux}}$, where g_1^s is a public parameter in pk_{ACC}. Then CA samples an identifier $ID_1 \leftarrow \mathbb{Z}_p$ and computes $F_{ID}(X) = X + ID_1$. Next, CA generates a signature σ for the pair (pk_u, C_1) by $Sign(sk, A_1, k', pk_U, ID_1)$.

- $Setup(1^\lambda, 1^n)$: $pp_\Sigma \leftarrow \Sigma.PPGen(1^\lambda, 1^n)$, $pp_{zk} \leftarrow ZKPok.Setup(1^\lambda)$, output $pp = (pp_\Sigma, pp_{zk})$.
- $CAKeyGen(pp, 1^l)$: CA runs $(sk_{CA}, pk_{CA}) \leftarrow \Sigma.KeyGen$, and outputs (sk_{CA}, pk_{CA}).
- $RAKeyGen(pp, 1^t)$: RA runs $(pk_{ACC}, sk_{ACC}) \leftarrow Gen_{ACC}(pp, t)$ to initialize the accumulator, setting $rpk = pk_{ACC}$ and $rsk = sk_{ACC}$. The space of accumulated elements is also in $\mathbb{Z}_p^\star$.
- $UKeyGen(pp)$: The user runs $(sk_u, pk_u) \leftarrow \Sigma.UKeyGen(pp)$, and outputs (sk_u, pk_u).
- $NymGen(pp, pk_u)$: The user picks random $\phi, \delta \leftarrow \mathbb{Z}_p^\star$, computes $nym_u \leftarrow Random_{PK}(pk_u, \phi, \delta)$, sets $sk_{nym} = (\phi, \delta)$, and outputs the pairs (nym_u, sk_{nym}).

– 1) If CA allows U to continue to delegate, CA can restrict the specific attributes and depth that U can proceed to sign by following the steps: CA first selects a random $t \leftarrow \mathbb{Z}_p$ and computes $C_{usign,j} \leftarrow g_1^t (g_1^{x_j'})^{\frac{1}{y}}$ for $2 \le j \le l$. Then CA sends it to RA, who in turn provides $E = (E_1, (E_{2,j})_{2 \le j \le l}) = (g_1^s, (C^s_{usign,j})_{2 \le j \le l})$. Finally, CA unblinds the usign: $usign_{j,2,1} \leftarrow (E_{2,j}/E_1^t) = ((g_1^{x_j'})^{\frac{s}{y}})$ for $j \in [2, l]$ and sets the delegation key as:

$$uk_{k'} = (usign_{j,1} usign_{j,2}), usign_{j,1} = ((l_i^{x_j})^{\frac{1}{y}})_{j \in [2,k'], i \in [I]}$$
$$usign_{j,2} = (usign_{j,2,1}, usign_{j,2,2}) = (((g_1^{x_j'})^{\frac{s}{y}}), ((g_1^{x_j'})^{\frac{1}{y}}))_{j \in [2,k']})$$

  where I is the set of attribute indexes that U can further delegate. CA sends $(\sigma, C_1, F_{ID}(X), uk_{k'})$ to U.
– 2) Else, CA just sends $(\sigma, C_1, F_{ID}(X))$ to U.
– U runs $(\sigma', C_1{}', (uk'_{k'} \text{ or } \perp), \delta, pk'_u) \leftarrow Random_{All}(pk_u, uk_{k'} \text{ or } \perp, C_1, \sigma, \mu, \phi)$ for μ, ϕ and stores $cred_U = \sigma'$, as well as $((uk'_{k'} \text{ or } \perp), C_1{}')$.

Delegate a credential: (by an issuer R and a user U). Corresponds the $DelegateCred/ReceiveCred$ interactive protocol. The details are as follows:

– U generates a pseudonym $(nym_U, sk_{nym}) \leftarrow NymGen(pp, pk_U)$ and a proof $\pi_{nym} \leftarrow ZKPok.Prove(sk_U, nym_U)$, then sends (nym_U, π_{nym}) along with an attributes vector A_L to R.
– R samples identifier $ID_L \leftarrow \mathbb{Z}_p$ and runs $(\sigma',\ \mathbf{C}',\ uk_{k''}) \leftarrow Delegate(A_L, uk_{k'}, \mathbf{C}, \sigma, k'', I_{L-1}, F_{ID}(X), ID_L)$, where $\mathbf{C}' = (\mathbf{C}, C_L)$. The identifier polynomial is updated to $F'_{ID}(X) = F_{ID}(X) \cdot (X + ID_L)$. Then R runs $\sigma_{pro} \leftarrow SendConvertSig(vk,\ sk_R,\ \sigma')$. If R allows U to continue to delegate, sends $(\sigma_{pro},\ \mathbf{C}',\ uk_{k''}, F'_{ID}(X))$ to U. Else, sends $(\sigma_{pro},\ \mathbf{C}',\ \perp, F'_{ID}(X))$ to U.
– U runs $\sigma' \leftarrow ReceiveConvertSig(vk, sk_U, \sigma_{pro})$ using σ_{pro} as input to covert the public key associated with the credential from pk_R to pk_U. Then U runs $(\sigma'', C'', (uk'_{k''} \text{ or } \perp), pk'_u, \delta) \leftarrow Random_{All}(pk_u, (uk_{k''} \text{ or } \perp), C', \sigma', \mu, \phi)$ using μ and ϕ, and stores $cred_U = \sigma''$ and $((uk'_{k''} \text{ or } \perp), C'')$.

Revoke a credential. When a user's credentials are found to have prohibited behaviors or delegation abilities revoked, RA can revoke his credentials. After a user's credential is revoked, all subsequent credentials issued based on that credential become invalid. The revocation information A_X are public, and users can obtain their non-membership proof in A_X from RA.

$ReceiveToken(F_{ID}(X)) \leftrightarrow CreateToken(F_{ID}(X), A_X, pp) \rightarrow \bar{\pi}$: RA runs $\bar{\pi} \leftarrow Acc.NonMemProve(A_X, F_{ID}(X))$, where $\bar{\pi} = (g_2^{\alpha(s)}, g_1^{\beta(s)}) \in \mathbb{G}_2 \times \mathbb{G}_1$. Then RA sends $\bar{\pi}$ to U.

$Revoke(pp, sk_{ACC}, ID) \rightarrow A'_X$: RA runs $A'_X \leftarrow Acc.add(A_X, ID, sk_{ACC}, pk_A)$ to update the accumulator, and publishes A'_X to the public.

Show a credential (by a user U and a verifier V). Correspond to the $Show/Verify$ interactive protocol. Assuming U positioned at layer L of the delegation chain and the verification rule Φ being public.

- U generates a pseudonym $(nym_U, sk_{nym}) \leftarrow NymGen(pp, pk_U)$ and a proof $\pi_{nym} \leftarrow ZKPok.Prove(sk_U, nym_U)$. Then, U samples $\mu \leftarrow \mathbb{Z}_p$ and runs $(\sigma', \mathbf{C}', \mathbf{W}', pk'_U, uk_{k'}, \delta) \leftarrow Random_{All}(pk_U, uk_{k'}, \mathbf{C}, \mathbf{W}, \sigma, \mu, \phi)$, where σ is regarded as $cred_U$.
- Selective disclosure: U chooses a subvector of attributes $A'_L \subset A_L$ that $\Phi(A'_L) = 1$ and runs $\pi_1 \leftarrow VC.Prove(pp, A_L, D, A'_L)$, where D is the set of attribute indexes of A'_L. U computes π'_1 by running $VC.Random(C_L, \pi_1, \mu)$.
- Prove that all credentials in the delegation chain have not been revoked: U computes an accumulator A_{ID} by running $Commit_{ACC}(F_{ID}(X), pp)$ and computes membership proofs $(w_i)_{i\in[L]}$ for the credential chain by executing $w_i \leftarrow Acc.MemProve(F_{ID}(X), C_{i,2})$ for $C_i = (C_{i,1}, C_{i,2})_{i\in[L]}$. U also sends $F_{ID}(X)$ to RA to get nonmembership proof $\bar{\pi}$.
- To ensure unlinkability, U re-randomizes $\bar{\pi}$ and $(w_i)_{i\in L}$ by selecting $\rho \leftarrow \mathbb{Z}_p$ and computing $(w_i)'_{i\in L} = (w_i)^{\rho}_{i\in L}$ and $\bar{\pi}' = ((g_2^{\alpha(s)'}), (g_1^{\beta(s)'})) = ((g_2^{\alpha(s)})^{\mu+r}, (g_1^{\beta(s)})^{-\rho})$. U randoms A_{ID} to $A'_{ID} = A_{ID}^{(\mu+r)\rho}$.

$$(nym_U, \pi_{nym}, \pi'_1, \sigma', \mathbf{C}, A'_L, \bar{\pi}', (w_i)'_{i\in L}, g_2^{\mu}, A'_{ID})$$

V accepts if all the following conditions are satisfied:
- π_{nym} is valid: $ZKPok.Verify(\pi_{nym}, nym_U) = 1$.
- π'_1 is valid: $VC.Verify(pp, C'_L, A'_L, D, \pi'_1, g_2^{\mu}) = 1$
- σ' is valid: $Verify(vk, nym_U, \mathbf{C}', \sigma') = 1$.
- $\bar{\pi}'$ is valid: $NonMemVerify_{ACC}(A_X, A'_{ID}, \bar{\pi}', PK_{ACC}) = 1$.
- w_i is valid: $MemVerifyPos_{ACC}(A'_{ID}, C'_{i,2}, w'_i, PK_{ACC}) = 1$ for $i \in L$.

Compactness of credentials. We claim that the credentials in DAAC-CR are constant-size since the SPSEQ-VC signature is constant-size. While delegation key depends on the depth to which the holder can delegate, it only affects those with delegation capabilities, and end-users, who are the majority of the system, are not bothered by these expenses.

Theorem 3. *If the Accumulator, structure-preserving signatures on Vector Commitment, and ZKPok are correct and unforgeability, The DAC-CR construction is correct, anonymity and unforgeability.*

The detailed proof is in the appendix C.

6 Implementation and Performance Evaluation

6.1 Experimental Evaluation

The experiments were conducted on a system equipped with an Intel Core Xeon(R) Platinum 8369HB CPU at 3.30GHz, running Ubuntu 20.04.6. We implemented the SPSEQ-VC and DAAC-CR prototypes in Python, utilizing the *bplib* library with *OpenSSL* bindings for cryptographic operations. The BN256 curve, known for its efficiency in Type-3 bilinear groups, was chosen to provide a security level of approximately 100 bits.

Our performance metrics focused on the time consumption of various operations within SPSEQ-VC and DAAC-CR, such as issuing, delegating, randomizing, and verifying credentials. We evaluated these metrics under varying conditions. In the experiments, the length of the commitment vector and the maximum delegation depth are fixed at 15 and 11, respectively.

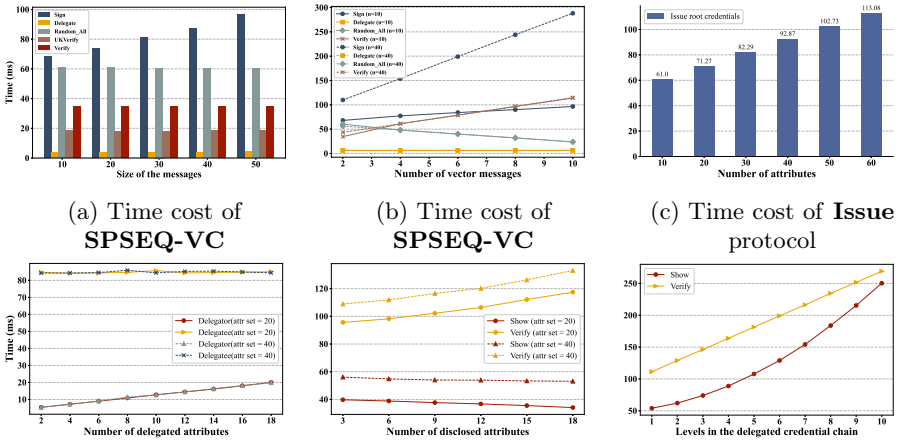
Evaluation on SPSEQ-VC. Figure 1a illustrates the impact on the computation time of SPSEQ-VC when the depth of the delegation chain (parameter k) is held constant while the size of the commitment vector (parameter n) is increased from 10 to 50. The results indicate that the computation time for the Sign algorithm increases with n. However, the computation times for other algorithms remain largely unaffected.

Figure 1b explores the effects on computation time when n is held constant at two levels, 10 and 40, and k is varied from 2 to 10. The findings demonstrate that increases in k notably affect the computation times of both the Sign and Verify algorithms. This suggests that deeper delegation chains tend to increase the computational load for these operations.

DAAC-CR.Issue Figure 1c illustrates the effect of increasing the number of attributes (parameter m) on the computation time of the *Issue* protocol while keeping the depth of the delegation chain (parameter k) fixed. The results show a consistent increase in computation time as m increases. This suggests that more attributes contribute to longer processing times.

DAAC-CR.Delegate Figure 1d examines the *Delegate* protocol's computation times as the number of delegated attributes (parameter d) increases. Interestingly, while the computation time for the issuer (Delegator) increases, the computation time for the user (Delegatee) remains relatively unaffected, regardless of the number of attributes each possesses.

DAAC-CR.Show/Verify Figures 1e and 1f explore the computation times involved in the $Show/Verify$ processes under varying conditions. The results show that while the $Show$ time slightly decreases, the $Verify$ time increases. The decrease in computation time observed in the $Show$ process can be attributed to



(a) Time cost of **SPSEQ-VC**

(b) Time cost of **SPSEQ-VC**

(c) Time cost of **Issue** protocol

(d) Time cost of **Delegate** protocol

(e) Time cost of **Show**/**Verify** protocol

(f) Time cost of **Show**/**Verify** protocol

Fig. 1: The results of our implementation for **SPSEQ-VC** and **DAAC-CR**

the method of polynomial division used during the vector commitment openings. Specifically, the use of larger subvectors results in a lower degree of the quotient polynomial, which in turn speeds up the generation of proofs. In Figure 1f, m was kept at 20 and s at 5, while k (the user's level in the delegation chain) ranged from 1 to 10. The computation times for both $Show$ and $Verify$ increased with higher k. This increase is attributed to the growth in the number of set membership proofs required and the size of the commitment vector as k rises.

Overall Efficiency. The implemented DAAC-CR schemes show high efficiency across various configurations. For instance, both $CreateCred$ and $DelegateCred$ can process attribute vectors of less than 50 in under 100 ms. During the credential presentation phase, even with attribute vectors not exceeding 20 attributes and a maximum of 5 selectively disclosed within a delegation chain of no more than 10 levels, the computation times for both $Show$ and $Verify$ remain below 300 ms. This performance is well-suited to handle the demands of most real-world applications, demonstrating the practical viability of the DAAC-CR schemes.

7 Conclusion

In this paper, we explore a novel approach for constructing DAAC schemes with fine-grained delegation constraints and flexible revocation. To this end, we first introduce a new concept, DAAC-CR, and formalize its syntax and security properties. Then, we propose a novel cryptographic primitive SPSEQ-VC, which extends SPSEQ-UC to support the specified opening messages and is well suited to the commit-ahead-of-time system. In addition, SPSEQ-VC provides an inter-

esting mapping in updating keys to support fine-grained adaptation. Moreover, we provide an efficient DAAC-CR instance using SPSEQ-VC, which is formally proven to be secure. Our scheme reaches the optimal credential size. The *delegation* in our DAAC-CR instance only takes 113 ms on 50 attributes, and both *Show* and $Verify$ remain below 300 ms for the 10-level user with 5 selectively disclosed attributes.

Acknowledges.

This work was supported by the Shenzhen Science and Technology Major Project (KJZD20230923114908017), the Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies (2022B1212010005), the National Natural Science Foundation of China (No. 62202375), the Shaanxi Distinguished Youth Project (2022JC-47), the Young Talent Fund of Association for Science and Technology in Shaanxi, China (No. 20220134) and the Major Program of Shandong Provincial Natural Science Foundation for the Fundamental Research (ZR2022ZD03).

A Additional Preliminaries

A.1 Collision Freeness.

An accumulator is said to be collision-free, if for all PPT adversaries $\mathcal{A}$ having oracle access to $\mathcal{O}$, security parameters λ there is a negligible function ϵ such that:

$$\Pr\left[\begin{matrix}(sk_{ACC}, pk_{ACC}) \leftarrow Gen_{ACC}(BG(t), (\omega_{x_i}, x_i, X, r) \leftarrow A^{\mathcal{O}}(pk_{ACC}) : \\ Verify_{ACC}(pk_{ACC}, A_X, \omega_{x_i}, x_i) = 1 \wedge x_i \in X\end{matrix}\right] \leq \varepsilon(\lambda),$$

B Proofs of SPSEQ-VC

B.1 Origin-hiding of SPSEQ-VC

Proof. Let $\mathbf{C}, \sigma = (Z, Y, Y', T)$ be such that $Verify(vk, \mathbf{C}, \sigma, pk_u, \mathbf{D}, \mathbf{W}) = 1$ along with $uk_{k'}$. For $\mu, \phi \leftarrow \mathbb{Z}_p$, $Random_{All}$ outputs $(\mathbf{C}', \sigma', pk'_u, uk'_{k'})$. Because these elements are randomly selected, all outputs of $Random_{All}$ are perfectly randomized and satisfy $Verify(vk, \mathbf{C}', \sigma', pk'_u, \mathbf{D}, \mathbf{W}) = 1$. Therefore, $Random_{All}$ apparently produces signatures of the same distribution as $Sign_{root}$.

B.2 Conversion-privacy of SPSEQ-VC

Proof. For all $(vk, sk) \in KeyGen(pp)$, $(sk_u, pk_u) \in UKeyGen(pp)$, $\mathbf{C}, \mathbf{M}, \mathbf{D}, \mathbf{W}$ and σ. If $Verify(vk, pk_u, \mathbf{C}, \sigma, \mathbf{D}, \mathbf{W}) = 1$. The signature σ:

$$\sigma = (Z \leftarrow (\prod_{i \in [k]} C_{i,1}^{x_i} C_{i,2}^{x'_i})^{\frac{1}{y}}, Y \leftarrow g_1^y, Y' \leftarrow g_2^y, T \leftarrow g_1^{x_0 \cdot y} \cdot pk_u^x)$$

Then for $(sk_{u'}, pk_{u'}) \in UKeyGen(pp)$, $ConvertSig(vk, sk_u, sk_{u'}, \sigma)$ outputs a new signature

$$\sigma' = (Z \leftarrow (\prod_{i\in[k]} C_{i,1}^{x_i} C_{i,2}^{x'_i})^{\frac{1}{y}}, Y \leftarrow g_1^y, Y' \leftarrow g_2^y, T \leftarrow g_1^{x_0 \cdot y} \cdot pk_{u'}^x)$$

It is clear that the output of $ConvertSig$ is distributed the same as the output of $Sign_{root}$.

B.3 Derivation-privacy of SPSEQ-VC

Proof. For all $(vk, sk) \in KeyGen(pp)$, pk_u, $\mathbf{M}$, $\mathbf{C}$, k', k'', $uk_{k'}, \mathbf{D}, \mathbf{W}$ and σ. If $Verify(vk, pk_u, \mathbf{C}, \sigma, \mathbf{D}, \mathbf{W}) = 1$, then for an index $L = k + 1 \in [k + 1, k']$, let M_L be a message set such that the message vector is $\mathbf{M}' = (\mathbf{M}, M_L)$ and the related commitment vector is $\mathbf{C}' = (\mathbf{C}, C_L)$. It is clear that $Delegate$ outputs the same distribution as $Sign_{root}$ for vectors $\mathbf{M}'$ and $\mathbf{C}'$: $Sign_{root}(sk, \mathbf{M}', k', pk_u) \approx Delegate(M_L, \sigma, \mathbf{C}, uk_{k'}, k'')$. And $ConvertSig$ outputs the same distributed as $Sign_{root}$.

B.4 Unforgeability of SPSEQ-VC

We prove that our scheme satisfies the unforgeability of ("Type-3") bilinear groups in the general group model (GGM) [26]. In this model, the adversary is only given handles of group elements, which are just uniform random strings. To execute group operations, the adversary utilizes an oracle that accepts handles as input and returns the handle corresponding to the sum, inversion, or other operations performed on the group elements associated with the provided handles. The experiment on unforgeability is as follows

$$\begin{aligned}
&EXPUnf_{\mathcal{A}}^{SPSEQ-VC}(1^\lambda, 1^l, 1^n, u) : \\
&u \leftarrow \mathcal{A}(); \\
&Q := \emptyset; \mathcal{UL} := \emptyset, pp \leftarrow PPGen(1^\lambda, 1^l, 1^n, u); \\
&\mathcal{UL} \leftarrow UKeyGen(pp); \\
&(vk, sk) \leftarrow KeyGen(pp); \\
&((sk_u^\star, pk_u^\star), \mathbf{C}^\star, \sigma^\star) \leftarrow \mathcal{A}^{\mathcal{O}^{Sign}}(vk, pp, \{pk_u\}_{(pk_u, sk_u)\in\mathcal{UL}}); \\
&\text{Return } \forall (pk_u, sk_u) \notin \mathcal{UL}, \forall (\mathbf{M}, k', pk_u) \in Q : \\
&(\mathbf{M}, \mathbf{D}) \notin \mathcal{R}_{k'} \wedge Verify(vk, pk_u^\star, \mathbf{C}^\star, \sigma^\star, \mathbf{D}^\star, \mathbf{U}^\star) = 1
\end{aligned}$$

$\mathcal{O}^{Sign}(\mathbf{M}, k', pk_u)$: If $k \leq k' \leq l$, then runs $(\sigma, \mathbf{C}, uk_{k'}) \leftarrow Sign_{root}(sk, k', \mathbf{M}, pk_u)$ and sets $Q = Q \cup (\mathbf{M}, k', pk_u)$. Return $(\mathbf{C}, \sigma, uk_{k'})$. Else, return $\perp$.

Proof. We examine an adversary in the game described above that solely employs generic group operations on the received group elements. After getting the public

parameter $(Pa^o, P'a^o)_{0\leq o\leq n}$, verification key $((\bar{X}, X, X_0, ..., X_l), (X'_1, ...X'_l)) = ((Px, P'x, P'x_0, ..., P'x_l), (P'x'_1, ..., P'x'_l))$, public keys $(P_1, ..., P_b)$, vector commitments $((C_1^{(i)}, .., C_{k^{(i)}}^{(i)}) = ((C_{1,1}^{(i)}, C_{1,2}^{(i)}), .., (C_{k^{(i)},1}^{(i)}, C_{k^{(i)},2}^{(i)})))_{i=1}^{q}$ and signatures $(Z_i, Y_i, Y'_i, T_i)_{i=1}^{q}$ computed with randomness y_i on queries $((M_1^{(i)}, ..., M_{k^{(i)}}^{(i)}), k^{(i)'}, pk^{(i)})_{i=1}^{q}$, delegation keys $(((U_{j,o}^{(i)}) = ((U_{j,o,1}^{(i)}, (U_{j,o,2}^{(i)}) = (U_{j,o,2,1}^{(i)}, U_{j,o,2,2}^{(i)}))_{j\in[k^{(i)}+1,k^{(i)'}],o\in[n]})_{i=1}^{q}$, the adversary outputs a user public key $pk^{(q+1)}$, a vector of commitments $(C_1^{(\star)}, ..., C_{k^{(\star)}}^{(\star)})$ and a corresponding signature $(Z^\star, Y^\star, Y^{\star'}, T^\star)$. As the adversary needs to compute new group elements by combining the received group elements, it is required to select coefficients represented by Greek letters, which define:

$$C_{h,1}^\star = \zeta^{(h)}P + \sum_{j=1}^{q}(\zeta_{z,j}^{(h)}Z_j + \zeta_{s,j}^{(s)}Y_j + \zeta_{n,j}^{(h)}T_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\zeta_{m,j,o,1}^{(h)}U_{m,o,1}^{(j)}$$
$$+\sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\zeta_{m,j,o,2,1}^{(h)}U_{m,o,2,1}^{(j)} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\zeta_{m,j,o,2,2}^{(h)}U_{m,o,2,2}^{(j)})$$
$$+\sum_{o=1}^{n}\zeta_{a,o}^{(h,q+1)}a^oP + \zeta_x^{(h)}\bar{X} + \sum_{u=1}^{b}\zeta_{pk,u}^{(h)}P_u$$

$$C_{h,2}^\star = \sigma^{(h)}P + \sum_{j=1}^{q}(\sigma_{z,j}^{(h)}Z_j + \sigma_{s,j}^{(s)}Y_j + \sigma_{n,j}^{(h)}T_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\sigma_{m,j,o,1}^{(h)}U_{m,o,1}^{(j)}$$
$$+\sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\sigma_{m,j,o,2,1}^{(h)}U_{m,o,2,1}^{(j)} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\sigma_{m,j,o,2,2}^{(h)}U_{m,o,2,2}^{(j)})$$
$$+\sum_{o=1}^{n}\sigma_{a,o}^{(h,q+1)}a^oP + \sigma_x^{(h)}\bar{X} + \sum_{u=1}^{b}\sigma_{pk,u}^{(h)}P_u$$

$$Z^\star = \kappa P + \sum_{j=1}^{q}(\kappa_{z,j}Z_j + \kappa_{s,j}Y_j + \kappa_{n,j}T_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\kappa_{m,j,o,1}U_{m,o,1}^{(j)})$$
$$+\sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\kappa_{m,j,o,2,1}^{(h)}U_{m,o,2,1}^{(j)} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\kappa_{m,j,o,2,2}^{(h)}U_{m,o,2,2}^{(j)})$$
$$+\sum_{o=1}^{n}\kappa_{a,o}a^oP + \kappa_x\bar{X} + \sum_{u=1}^{b}\kappa_{pk,u}^{(h)}P_u$$

$$S^\star = \nu P + \sum_{j=1}^{q}(\nu_{z,j}Z_j + \nu_{s,j}Y_j + \nu_{n,j}T_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \nu_{m,j,o,1}U^{(j)}_{m,o,1})$$
$$+ \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \nu^{(h)}_{m,j,o,2,1}U^{(j)}_{m,o,2,1} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \nu^{(h)}_{m,j,o,2,2}U^{(j)}_{m,o,2,2})$$
$$+ \sum_{o=1}^{n}\nu_{a,o}a^oP + \nu^{(h)}_x\bar{X} + \sum_{u=1}^{b}\nu^{(h)}_{pk,u}P_u$$

$$T^\star = \rho P + \sum_{j=1}^{q}(\rho_{z,j}Z_j + \nu_{s,j}Y_j + \rho_{n,j}T_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \rho_{m,j,o,1}U^{(j)}_{m,o,1})$$
$$+ \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \mu^{(h)}_{m,j,o,2,1}U^{(j)}_{m,o,2,1} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \mu^{(h)}_{m,j,o,2,2}U^{(j)}_{m,o,2,2})$$
$$+ \sum_{o=1}^{n}\rho_{a,o}a^oP + \rho_x\bar{X} + \sum_{u=1}^{b}\rho^{(h)}_{pk,u}P_u$$

$$pk^{(h)} = \eta^{(h)}P + \sum_{j=1}^{q}(\eta^{(h)}_{z,j}Z_j + \eta^{(s)}_{s,j}Y_j + \eta^{(h)}_{n,j}T_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta^{(h)}_{m,j,o,1}U^{(j)}_{m,o,1})$$
$$+ \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta^{(h)}_{m,j,o,2,1}U^{(j)}_{m,o,2,1} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta^{(h)}_{m,j,o,2,2}U^{(j)}_{m,o,2,2})$$
$$+ \sum_{o=1}^{n}\eta^{(h,q+1)}_{a,o}a^ogP + \eta^{(h)}_x\bar{X} + \sum_{u=1}^{b}\eta^{(h)}_{pk,u}P_u$$

$$S^{\star\prime} = \epsilon P' + \epsilon_x X + \sum_{j=0}^{l}\epsilon_{x,j}X_j + \sum_{j=1}^{l}\epsilon_{x',j}X'_j + \sum_{j=0}^{q}\epsilon_{s,j}Y'_j + \sum_{o=1}^{t}\epsilon_{a,o}a^oP'$$

Using this, we can write, for all $1 \leq i \leq q$, the discrete logarithms $c^{(i)}_j, z_i$ and t_i in basis P of the elements $C^{(i)}_{j,1} = \prod_{e\in M^{(i)}_j} L_j eP$, $C^{(i)}_{j,2} = (s + ID_j)P$ $Z_i = \frac{\sum_{j=1}^{k^{(i)}} x_j C^{(i)}_{j,1} + x'_j C^{(i)}_{j,2}}{y_i}, T_i = X_0 y_i + xpk^{(i)}$ and $U^{(i)}_{m,o,1} = \frac{a^o x_m}{y_i}L_oP, U^{(i)}_{m,o,2} = (U^{(i)}_{m,o,2,1}, U^{(i)}_{m,o,2,2}) = (\frac{a^o x'_m s}{y_i}P, \frac{a^o x'_m}{y_i}P)$ from the oracle answers.

$$c^{(i)}_{j,1} = \prod_{e\in M^{(i)}_j} L_i e, c^{(i)}_{j,2} = \prod_{j\in k}(s + ID_j) \tag{1}$$

$$z_i = \frac{\sum_{j=1}^{k^{(i)}} x_j c^{(i)}_{j,1} + x'_j c^{(i)}_{j,2}}{y_i} \tag{2}$$

$$t_i = x_0 y_i + xpk^{(i)} \tag{3}$$

$$u_{m,o}^{(i)} = (\frac{a^o x_m}{y_i} L_o, (\frac{a^o x'_m s}{y_i}, \frac{a^o x'_m}{y_i})) \tag{4}$$

A successful forgery $(Z^\star, Y^\star, Y^{\star'}, T\star)$ on $(pk^{(q+1)}, C_1^\star, ..., C_{k^{(q+1)}}^\star)$ satisfies the verification equations

$$e(Z^\star, S^{\star'}) = \prod_{h=1}^{k^{(q+1)}} e(C_{h,1}^\star, X_h) \cdot e(C_{h,2}^\star, X'_h)$$
$$\wedge\, e(P, S^{\star'}) = e(S^\star, P') \wedge e(T^\star, P') = e(S^\star, X_0) \cdot e((pk^{(q+1)}, X)$$

We interpret these values as multivariate rational fractions in variables $x, x_0, ..., x_l, x'_1, ..., x'_l, y_1, ..., y_q, a, p_1, ..., p_b$. Using the coefficients defined above and considering the logarithms in base $e(P, P')$ we obtain:

$$\begin{aligned}
&(\kappa + \sum_{j=1}^{q}(\kappa_{z,j} z_j + \kappa_{s,j} y_j + \kappa_{n,j} t_j + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \kappa_{m,j,o,1} u_{m,o,1}^{(j)}) \\
&+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \kappa_{m,j,o,2,1}^{(h)} u_{m,o,2,1}^{(j)} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \kappa_{m,j,o,2,2}^{(h)} u_{m,o,2,2}^{(j)}) \\
&+ \sum_{o=1}^{n} \kappa_{a,o} a^o + \kappa_x x + \sum_{u=1}^{b} \kappa_{pk,u}^{(h)} p_u) \cdot \\
&(\epsilon + \epsilon_x x + \sum_{j=0}^{l} \epsilon_{x,j} x_j + \sum_{j=1}^{l} \epsilon_{x',j} x'_j + \sum_{j=1}^{q} \epsilon_{s,j} y_j + \sum_{o=1}^{t} \epsilon_{a,o} a^o) = \sum_{h=1}^{k^{(q+1)}} (x_h c_{h,1}^\star + x'_h c_{h,2}^\star)
\end{aligned} \tag{5}$$

$$\begin{aligned}
&\nu + \sum_{j=1}^{q}(\nu_{z,j} z_j + \nu_{s,j} y_j + \nu_{n,j} t_j + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu_{m,j,o,1} u_{m,o,1}^{(j)}) \\
&+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu_{m,j,o,2,1}^{(h)} u_{m,o,2,1}^{(j)} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu_{m,j,o,2,2}^{(h)} u_{m,o,2,2}^{(j)}) \\
&+ \sum_{o=1}^{n} \nu_{a,o} a^o + \nu_x^{(h)} x + \sum_{u=1}^{b} \nu_{pk,u}^{(h)} p_u \\
&= \epsilon + \epsilon_x x + \sum_{j=0}^{l} \epsilon_{x,j} x_j + \sum_{j=1}^{l} \epsilon_{x',j} x'_j + \sum_{j=1}^{q} \epsilon_{s,j} y_j + \sum_{o=1}^{t} \epsilon_{a,o} a^o
\end{aligned} \tag{6}$$

$$
\begin{aligned}
&\rho + \sum_{j=1}^{q}(\rho_{z,j} z_j + \nu_{s,j} y_j + \rho_{n,j} t_j + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \rho_{m,j,o,1} u_{m,o,1}^{(j)}) \\
&+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \rho_{m,j,o,2,1}^{(h)} u_{m,o,2,1}^{(j)} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \rho_{m,j,o,2,2}^{(h)} u_{m,o,2,2}^{(j)}) \\
&+ \sum_{o=1}^{n} \rho_{a,o} a^o + \rho_x x + \sum_{u=1}^{b} \rho_{pk,u}^{(h)} p_u \\
&= x_0(\nu + \sum_{j=1}^{q}(\nu_{z,j} z_j + \nu_{s,j} y_j + \nu_{n,j} t_j + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \nu_{m,j,o,1} u_{m,o,1}^{(j)}) \\
&+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \nu_{m,j,o,2,1}^{(h)} u_{m,o,2,1}^{(j)} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \nu_{m,j,o,2,2}^{(h)} u_{m,o,2,2}^{(j)}) \\
&+ \sum_{o=1}^{n} \nu_{a,o} a^o + \nu_x^{(h)} x + \sum_{u=1}^{b} \nu_{pk,u}^{(h)} p_u \\
&+ x(\eta^{(q+1)} + \sum_{j=1}^{q}(\eta_{z,j}^{(q+1)} z_j + \eta_{s,j}^{(q+1)} y_j + \eta_{n,j}^{(q+1)} t_j + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta_{m,j,o,1}^{(q+1)} u_{m,o,1}^{(j)}) \\
&+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta_{m,j,o,2,1}^{(q+1)} u_{m,o,2,1}^{(j)} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta_{m,j,o,2,2}^{(q+1)} u_{m,o,2,2}^{(j)}) \\
&+ \sum_{o=1}^{n} \eta_{a,o}^{(h,q+1)} a^o g + \eta_x^{(q+1)} x + \sum_{u=1}^{b} \eta_{pk,u}^{(q+1)} p_u)
\end{aligned}
\tag{7}
$$

To apply the standard proof technique in the generic group model, we proceed by considering an "ideal" game scenario. In this ideal game, the challenger treats all the handles of group elements as elements of $\mathbb{Z}_p(y_1, ..., y_q, x, x_0, ..., x_l, x'_1, ..., x'_l, a, p_1, ..., p_b)$. Here, the elements are rational fractions, and the indeterminates represent the secret values chosen by the challenger.

Initially, we demonstrate in the game that if the adversary's output satisfies the verification equations, the first winning condition is not fulfilled. This indicates that winning the game is not possible.

We thus interpret the three equalities (5), (6) and (7) as polynomial equalities over the field $\mathbb{Z}_p(y_1, ..., y_q, x, x_0, ..., x_l, x'_1, ..., x'_l, a, p_1, ..., p_b)$. More precisely, we consider the equalities in the ring $\mathbb{Z}_p(y_1, ..., y_q)[x, x_0, ..., x_l, x'_1, ..., x'_l, a, p_1, ..., p_b]$, that is, the polynomial ring with $x, x_0, ..., x_l, x'_1, ..., x'_l, a, p_1, ..., p_b$ as indeterminates over the field $\mathbb{Z}_p(y_1, ..., y_q)$. As one of our proof techniques, we will also consider the equalities over the ring

factored by $(x, x_0, ..., x_l, x'_1, ..., x'_l)$, the ideal generated by the $x'_i s$:

$$\begin{aligned}&\mathbb{Z}_p(y_1, ..., y_q), [x, x_0, ..., x_l, x'_1, ..., x'_l, a, p_1, ..., p_b]/(x, x_0, ..., x_l, x'_1, ..., x'_l)\\ &\cong \mathbb{Z}_p(y_1, ..., y_q), [a, p_1, ..., p_b]\end{aligned}$$

From (2) and (3), over this quotient we have and $t_i = z_i = u^{(j)}_{m,o,1} = u^{(j)}_{m,o,2,1} = u^{(j)}_{m,o,2,1} = 0$ and thus (5)–(7) become

$$\begin{aligned}&(\kappa + \sum_{j=1}^{q} \kappa_{s,j} y_j + \sum_{o=1}^{n} \kappa_{a,o} a^o + \sum_{u=1}^{b} \kappa^{(h)}_{pk,u} p_u)\\ &(\epsilon + \sum_{j=1}^{q} \epsilon_{s,j} y_j + \sum_{o=1}^{n} \epsilon_{a,o} a^o) = 0\end{aligned} \tag{8}$$

$$\begin{aligned}&\nu + \sum_{j=1}^{q} \nu_{y,j} y_j + \sum_{o=1}^{n} \nu_{a,o} a^o + \sum_{u=1}^{b} \nu^{(h)}_{pk,u} p_u\\ &= \epsilon + \sum_{j=1}^{q} \epsilon_{y,j} y_j + \sum_{o=1}^{n} \epsilon_{a,o} a^o\end{aligned} \tag{9}$$

$$\rho + \sum_{j=1}^{q} \nu_{s,j} y_j + \sum_{o=1}^{n} \rho_{a,o} a^o + \sum_{u=1}^{b} \rho^{(h)}_{pk,u} p_u = 0 \tag{10}$$

By looking the coefficients of the monomials $s'_j s$, $p'_u s$, $a^{o'} s$ and 1 of (10), we deduce:

$$\forall j, o, u : \rho = \rho_{s,j} = \rho_{a,o} = \rho_{pk,u} = 0 \tag{11}$$

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

$$\begin{aligned}&\nu = \epsilon, \forall o : \nu_{a,o} = \epsilon_{a,o},\\ &\forall j : \nu_{s,j} = \epsilon_{s,j}\\ &\forall u : \nu_{pk,u} = 0\end{aligned} \tag{12}$$

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

$$\begin{aligned}&\sum_{j=1}^{q} (\nu_{z,j} z_j + \nu_{n,j} s_j x_0 + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu_{m,j,o,1} u^{(j)}_{m,o,1})\\ &+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu^{(h)}_{m,j,o,2,1} u^{(j)}_{m,o,2,1} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu^{(h)}_{m,j,o,2,2} u^{(j)}_{m,o,2,2}) \ mod\ (x)\\ &= \sum_{j=0}^{l} \epsilon_{x,j} x_j + \sum_{j=1}^{l} \epsilon_{x',j} x'_j\end{aligned} \tag{13}$$

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

$$\forall j \geq 0 : \epsilon_{x,j} = \epsilon_{x',j} = 0 \tag{14}$$

Then the equation (13) becomes:

$$\begin{aligned} &\sum_{j=1}^{q}(\nu_{z,j} z_j + \nu_{n,j} s_j x_0 + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu_{m,j,o,1} u^{(j)}_{m,o,1}) \\ &+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu^{(h)}_{m,j,o,2,1} u^{(j)}_{m,o,2,1} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu^{(h)}_{m,j,o,2,2} u^{(j)}_{m,o,2,2}) = 0 \end{aligned} \tag{15}$$

By looking all the monomials in $x_0 y_j$, we deduce:

$$\forall j : \nu_{n,j} = 0 \tag{16}$$

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

$$\begin{aligned} \forall j : (&\nu_{z,j} \frac{\sum_{m=1}^{k^{(m)}} x_m c^{(j)}_{m,1} + x'_m c^{(m)}_{m,2}}{y_j} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu_{m,j,o,1} \frac{a^o x_m}{y_j} L_j \\ &+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu^{(h)}_{m,j,o,2,1} \frac{a^o x'_m}{y_j} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)'}} \nu^{(h)}_{m,j,o,2,2} \frac{a^o x'_m}{y_j}) = 0 \end{aligned} \tag{17}$$

Then, by looking the monomials in $a^o x_j, a^o x'_j$ for all $j > k^{(j)}$:

$$\forall m, j, o : \nu_{m,j,o,1} = 0 = \nu_{m,j,o,2,1} = 0 = \nu_{m,j,o,,2,1} = 0 \tag{18}$$

Then (17) becomes:

$$\forall j : \nu_{z,j} \frac{\sum_{m=1}^{k^{(m)}} x_m c^{(j)}_{m,1} + x'_m c^{(m)}_{m,2}}{y_j} = 0 \tag{19}$$

Then without any loss of generalities, we deduce:

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

Now we look the equation (5) modulo $(x_0, x_1, ..., x_l, x'_1, ..., x'_l)$:

$$\begin{aligned} &(\kappa + \sum_{j=1}^{q}(\kappa_{s,j} y_j + \kappa_{n,j} t_j) + \sum_{o=1}^{n} \kappa_{a,o} a^o + \kappa_x x + \sum_{u=1}^{b} \kappa^{(h)}_{pk,u} p_u) \cdot \\ &(\epsilon + \epsilon_x x + \sum_{j=1}^{q} \epsilon_{s,j} y_j + \sum_{o=1}^{n} \epsilon_{a,o} a^o) \ mod \ (x_0, x_1, ..., x_l, x'_1, ..., x'_l) = 0 \end{aligned} \tag{21}$$

Now, because $S^{\star'} \neq 0$, we can deduce: $(\epsilon + \epsilon_{x,0}x_0 + \sum_{j=1}^{q} \epsilon_{s,j}y_j + \sum_{o=1}^{n} \epsilon_{a,o}a^o) \neq 0$, then because

$$t_j \ mod \ (x_0, x_1, ..., x_l, x'_1, ..., x'_l) = \frac{x}{s_j} pk^{(j)} \ mod \ (x_0, x_1, ..., x_l, x'_1, ..., x'_l)$$

we have:

$$\begin{aligned} &(\kappa + \sum_{j=1}^{q}(\kappa_{s,j}y_j + \kappa_{n,j}\frac{x}{s_j}pk^{(j)}) + \sum_{o=1}^{n}\kappa_{a,o}a^o + \kappa_x x + \sum_{u=1}^{b}\kappa^{(h)}_{pk,u}p_u) \\ &mod \ (x_0, x_1, ..., x_l, x'_1, ..., x'_l) = 0 \end{aligned} \tag{22}$$

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

$$\begin{aligned} &\forall j : \kappa_{s,j} = 0, \\ &\forall o : \kappa_{a,o} = 0 \\ &\kappa = 0 \\ &\forall u : \kappa_{pk,u} = 0 \end{aligned} \tag{23}$$

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

$$\kappa_{x,0} = 0 \tag{24}$$

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

$$\begin{aligned} &\sum_{j=1}^{q}(\rho_{z,j}z_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\rho_{m,j,o,1}u^{(j)}_{m,o,1} \\ &+ \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\rho^{(h)}_{m,j,o,2,1}u^{(j)}_{m,o,2,1} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\rho^{(h)}_{m,j,o,2,2}u^{(j)}_{m,o,2,2}) \ mod \ (x, x_0) = 0 \end{aligned}$$

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

$$\begin{aligned} &\forall j : \rho_{z,j}\frac{\sum_{m=1}^{k^{(m)}} x_m c^{(j)}_{m,1} + x'_m c^{(m)}_{m,2}}{y_j} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\rho_{m,j,o}\frac{a^o x_m}{y_j}L_j \\ &+ \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\rho^{(h)}_{m,j,o,2,1}\frac{a^o x'_m s}{y_j} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\rho^{(h)}_{m,j,o,2,2}\frac{a^o x'_m}{y_j}) \ mod \ (x, x_0) = 0 \end{aligned} \tag{25}$$

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

$$\forall m, j, o : \rho_{m,j,o,1} = \rho_{m,j,o,2,1} = \rho_{m,j,o,,2,2} = 0 \tag{26}$$

Then (7) modulo (x) becomes:

$$\sum_{j=1}^{q}(\rho_{z,j}z_j + \rho_{n,j}x_0y_j) \ mod \ (x) = x_0(\nu + \sum_{j=1}^{q}\nu_{s,j}y_j + \sum_{o=1}^{n}\nu_{a,o}a^o) \tag{27}$$

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

$$\forall o : \nu = \nu_{a,o} = 0 \tag{28}$$

(6) becomes thus:

$$\sum_{j=1}^{q}(\nu_{z,j}z_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\nu_{m,j,o,1}u^{(j)}_{m,o,1} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\nu^{(h)}_{m,j,o,2,1}u^{(j)}_{m,o,2,1} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\nu^{(h)}_{m,j,o,2,2}u^{(j)}_{m,o,2,2}) + \nu^{(h)}_x x = \epsilon_x x \tag{29}$$

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

$$\nu_x = \epsilon_x \tag{30}$$

Then, by using (28),in (27), we deduce:

$$\sum_{j=1}^{q}(\rho_{z,j}z_j + \rho_{n,j}x_0y_j) \ mod \ (x) = x_0(\sum_{j=1}^{q}\nu_{s,j}y_j) \tag{31}$$

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

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

Then we can assume:

$$\forall j : \rho_{z,j} = 0 \tag{32}$$

Then by looking monomials in x_0y_j, for all j, deduce:

$$\forall j : \nu_{s,j} = \rho_{n,j} \tag{33}$$

Then, (7) becomes:

$$\sum_{j=1}^{q}(\rho_{n,j}t_j) + \rho_x x = x_0(\sum_{j=1}^{q}\nu_{s,j}y_j + \nu_x x) + xpk^{(q+1)} \tag{34}$$

Now, if we look the monomial xx_0, we deduce:

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

And (30) implies:

$$\epsilon_{x,0} = 0 \tag{36}$$

Now, let's look (5) by using (36), (14), (12):

$$\begin{aligned}
&(\sum_{j=1}^{q}(\kappa_{z,j}z_j + \kappa_{n,j}t_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\kappa_{m,j,o,1}u_{m,o,1}^{(j)}) \\
&+\sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\kappa_{m,j,o,2,1}^{(h)}u_{m,o,2,1}^{(j)} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\kappa_{m,j,o,2,2}^{(h)}u_{m,o,2,2}^{(j)}) \\
&(\sum_{j=1}^{q}\epsilon_{y,j}y_j) = \sum_{h=1}^{k^{(q+1)}}(x_h c_{h,1}^{\star} + x'_h c_{h,2}^{\star})
\end{aligned} \tag{37}$$

Let i_0 be the maximum of the $i's$ such that$\epsilon_i \neq 0$. Then $(\sum_{j=1}^{q}\epsilon_{y,j}y_j) = (\sum_{j=1}^{i_0}\epsilon_{y,j}y_j)$ is of degree 1 in y_{i_0} and in y_{i_1}.

Now we can notice that in $\sum_{h=1}^{k^{(q+1)}}(x_h c_{h,1}^{\star}+x'_h c_{h,2}^{\star})$, there is neither monomial of degree -1 in y_i and of degree 1 in y_k with $k \neq i$ nor monomials in $y_i y_k$, nor in y_i^2

Because, $(\sum_{j=1}^{i_0}\epsilon_{y,j}y_j)$ is of degree 1 in y_{i_0}. We deduce that the left term has no term of degree 1 or -1 in any indeterminate y_i in $y_1, ..., y_q$. In particular, there is no monomials in $x_1 y_i$ in this term, Then

$$\forall i : \kappa_{t,i}t_i = 0 \tag{38}$$

$$\begin{aligned}
\forall j \neq i_0 :&\kappa_{z,i_0}z_{i_0} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\kappa_{m,j,o,1}u_{m,o,1}^{(j)} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\kappa_{m,j,o,2,1}^{(h)}u_{m,o,2,1}^{(j)} \\
&+\sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)'}}\kappa_{m,j,o,2,2}^{(h)}u_{m,o,2,2}^{(j)}) = 0
\end{aligned} \tag{39}$$

Then (37) becomes:

$$\begin{aligned}
&(\kappa_{z,i_0} z_{i_0} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(i_0)'}} \kappa_{m,j,o,1} u_{m,o,1}^{(j)}) \\
&+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(i_0)'}} \kappa_{m,j,o,2,1}^{(h)} u_{m,o,2,1}^{(j)} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(i_0)'}} \kappa_{m,j,o,2,2}^{(h)} u_{m,o,2,2}^{(j)}) \\
&(\sum_{j=1}^{i_0} \epsilon_{y,j} y_j) = \sum_{h=1}^{k^{(q+1)}} (x_h c_{h,1}^{\star} + x'_h c_{h,2}^{\star})
\end{aligned} \tag{40}$$

By noticing $(\sum_{j=1}^{i_0} \epsilon_{y,j} y_j) = \sum_{h=1}^{k^{(q+1)}} (x_h c_{h,1}^{\star} + x'_h c_{h,2}^{\star})$ has no monomial in $y\frac{y_j}{y_{i_0}}$, for $j < i_0$, we deduce:

$$\forall j < i_0 : \kappa_{s,j} = 0 \tag{41}$$

We can now transform (40) in:

$$\begin{aligned}
&(\kappa_{z,i_0} \sum_{m=1}^{k^{i_0}} (x_m c_{m,1}^{i_0} + x'_m c_{m,2}^{i_0}) + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(i_0)'}} \kappa_{m,j,o,1} a^o x_m L_i \\
&+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(i_0)'}} \kappa_{m,j,o,2,1}^{(h)} a^o x'_m s y_i + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(i_0)'}} \kappa_{m,j,o,2,2}^{(h)} a^o x'_m y_i \\
&= \sum_{m=1}^{k^{(q+1)}} (x_m c_{m,1}^{\star} + x'_m c_{m,2}^{\star})
\end{aligned} \tag{42}$$

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

$$\forall m \leq k^{(i_0)} : \kappa_{z,i_0} c_{m,1}^{(i_0)} = c_{m,1}^{\star}, \kappa_{z,i_0} c_{m,2}^{(i_0)} = c_{m,2}^{\star} \tag{43}$$

$$\begin{aligned}
\forall m \in \left\{k^{(i_0)} + 1, ..., k^{'(i_0)}\right\} : \sum_{o=0}^{n} \kappa_{m,j,o} a^o L_i = c_{m,1}^{\star} \\
\sum_{o=0}^{n} \kappa_{m,j,o,2,1} a^o s y_i + \sum_{o=0}^{n} \kappa_{m,j,o,2,2} a^o y_i = c_{m,2}^{\star}
\end{aligned} \tag{44}$$

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

$$\begin{aligned}
&\rho_{y,i_0} t_{i_0} + \rho_x x \\
&= x_0 \nu_{y,i_0} t_{i_0} + x(\eta^{(q+1)} + \sum_{j=1}^{q} (\eta_{z,j}^{(q+1)} z_j + \eta_{y,j}^{(q+1)} y_j + \eta_{n,j}^{(q+1)} t_j + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta_{m,j,o,1}^{(q+1)} u_{m,o,1}^{(j)}) \\
&+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta_{m,j,o,2,1}^{(q+1)} u_{m,o,2,1}^{(j)} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta_{m,j,o,2,2}^{(q+1)} u_{m,o,2,2}^{(j)}) \\
&+ \sum_{o=1}^{n} \eta_{a,o}^{(h,q+1)} a^o g + \eta_x^{(q+1)} x + \sum_{u=1}^{b} \eta_{pk,u}^{(q+1)} p_u)
\end{aligned} \tag{45}$$

It becomes:

$$\begin{aligned}
&\rho_{y,i_0} x pk^{(i_0)} + \rho_x x \\
&= x_0 \nu_{y,i_0} t_{i_0} + x(\eta^{(q+1)} + \sum_{j=1}^{q} (\eta_{z,j}^{(q+1)} z_j + \eta_{y,j}^{(q+1)} y_j + \eta_{n,j}^{(q+1)} t_j + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta_{m,j,o,1}^{(q+1)} u_{m,o,1}^{(j)}) \\
&+ \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta_{m,j,o,2,1}^{(q+1)} u_{m,o,2,1}^{(j)} + \sum_{o=0}^{n} \sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \eta_{m,j,o,2,2}^{(q+1)} u_{m,o,2,2}^{(j)}) \\
&+ \sum_{o=1}^{n} \eta_{a,o}^{(h,q+1)} a^o g + \eta_x^{(q+1)} x + \sum_{u=1}^{b} \eta_{pk,u}^{(q+1)} p_u)
\end{aligned} \tag{46}$$

Then

$$\rho_{y,i_0} pk^{(i_0)} + \rho_x = pk^{(q+1)} \tag{47}$$

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

$$pk^{(i_0)} = \frac{(\eta - \rho_x)}{\rho_{y,i_0}} \tag{48}$$

We deduce that $pk^{(i_0)} \notin \mathcal{UL}$. It implies that $(\mathbf{M}_{i_0}, \mathbf{D}^\star) \notin \mathcal{R}_{k^{(i_0)}}$, thus $\exists j_0 \in 1, ..., k^{(i_0)}$, such that $D_{j_0} \not\subset M_{j_0}^{(i_0)}$. Then, it exists $e' \in D_{j_0} \backslash M_{j_0}^{(i_0)}$ Now, let's

look how $W_h^\star$ has been built by the adversary:

$$W_{j_0}^\star = \omega P + \sum_{j=1}^{q}(\omega_{z,j}Z_j + \omega_{y,j}Y_j + \omega_{n,j}T_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \omega_{m,j,o,1}U_{m,o,1}^{(j)})$$
$$+ \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \omega_{m,j,o,2,1}^{(h)}U_{m,o,2,1}^{(j)} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \omega_{m,j,o,2,2}^{(h)}U_{m,o,2,2}^{(j)})$$
$$+ \sum_{o=1}^{n}\omega_{a,o}a^oP + \omega_x\bar{X} + \sum_{u=1}^{b}\omega_{pk,u}^{(h)}P_u$$

Because $VC.Verify$ outputs 1 then:

$$(\omega + \sum_{j=1}^{q}(\omega_{z,j}z_j + \omega_{y,j}y_j + \omega_{n,j}t_j + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \omega_{m,j,o,1}u_{m,o,1}^{(j)})$$
$$+ \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \omega_{m,j,o,2,1}^{(h)}u_{m,o,2,1}^{(j)} + \sum_{o=0}^{n}\sum_{m=k^{(j)}+1}^{k^{(j)\prime}} \omega_{m,j,o,2,2}^{(h)}u_{m,o,2,2}^{(j)})$$
$$+ \sum_{o=1}^{n}\omega_{a,o}a^o + \omega_x x + \sum_{u=1}^{b}\omega_{pk,u}^{(h)}p_u) \cdot A_{D_{j_0}^\star} \tag{49}$$
$$+ (\prod_{e\in D_{j_0}^\star} L_i e) = \prod_{e\in M_{j_0}^{(i_0)}} L_i e$$

This equation subtract $L_j e'$ and modulo $(a - \omega^j)$:

$$(\prod_{e\in M_{j_0}^{(i_0)}} L_i e - L_j e') \ mod \ (a - \omega^j) = 0$$

Then, we have a contradiction, because $e' \notin D_{j_0}^{(i_0)}$. We have thus shown that the adversary cannot win the game.

C Security of DAAC-CR

(Unforgeability). If ZKPoK is a zero knowledge proof algorithm and SPSEQ-VC and accumulator scheme ACC are unforgeable, then the DAAC-CR construction in Section 5 is unforgeable.

Proof. Assume a PPT adversary $\mathcal{A}$ that wins the unforgeability game with non-negligible probability, then we can use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the unforgeability of SPSEQ-VC and accumulator. Now we can reduce the unforgeability from the following two steps:

Step1. We first prove that an adversary cannot propose a signature on a commitment vector open to a (sub)set of unsigned messages. $\mathcal{B}$ interacts with a challenger $\mathcal{C}$ in the unforgeability game for SPSEQ-VC and $\mathcal{B}$ simulates the DAAC-CR-unforgeability game for $\mathcal{A}$. $\mathcal{C}$ runs $(pp, sk_{CA}, pk_{CA}) \leftarrow Setup(1^{\lambda}, 1^{n}, 1^{l})$ and gives (pk_{CA}, pp) to $\mathcal{B}$. Then, $\mathcal{B}$ sends $pp, vk = pk_{CA}$ to $\mathcal{A}$. It next simulates the environment and oracles. All oracles are executed as in the real game, except for the following oracles, which use the signing oracle instead of using the signing key sk_{CA}:

When the oracles $\mathcal{O}^{CU}$ and $\mathcal{O}^{HU}$ are called, $\mathcal{B}$ queries $\mathcal{O}^{CU}$ and $\mathcal{O}^{HU}$ of the SPSEQ-UC scheme, respectively. Note that when $\mathcal{B}$ queries $\mathcal{O}^{CU}$ of the SPSEQ-UC, it gets sk_i and finally returns sk_i and all the associated credentials items to $\mathcal{A}$.

$\mathcal{O}^{RIO}(i, k', A_i)$: Takes as input a user identity i, an index k' and attributes A_i. If $i \in \mathcal{HU}$, returns $\perp$, else it submits (nym_i, k', A_i) to the signing oracle $\mathcal{O}^{Sign}$. Receives a signature $(\sigma = (Z, Y, Y', T), \mathbf{C}, uk_{k'})$. It sets $cred_i = (\sigma, \mathbf{C}, nym_i)$ and adds $(i, A_i, uk_{k'}, cred_i)$ to $\mathcal{L}_{cred}$.

$\mathcal{O}^{RIS}(k', A_i)$: On input an index k' and attributes A_i. it submits (nym_i, k', A_i) to the signing oracle, where nym_i is an adversary pseudonym of a corrupted user. Receives a signature $(\sigma = (Z, Y, Y', T), \mathbf{C}, uk_{k'})$. It sets $cred_i = (\sigma, \mathbf{C}, nym_i)$ and adds $(i, A_i, uk_{k'}, cred_i)$ to $\mathcal{L}_{cred}$ and outputs the results.

The oracles ($\mathcal{O}^{CIS}$, $\mathcal{O}^{OIS}$): In both $\mathcal{O}^{CIS}$, $\mathcal{O}^{OIS}$, all executions of *Delegate* and *ConvertSig* for credentials $(i, A_i, uk_{k'}, cred_i) \in \mathcal{L}_{cred}$ are replaced by the oracle $Sign(sk_{CA}, A_i, k', pk_i)$, where $pk_i = nym_i = 1$ for the $\mathcal{O}^{CIS}$ and $pk_i = nym_j$ for the $\mathcal{O}^{OIS}$.

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 $\mathbf{M}^{\star}$. To do this, $\mathcal{B}$ interacts with $\mathcal{A}$ as verifier in a showing protocol. If $\mathcal{A}$ outputs a valid showing proof as $(\mathbf{C}^{\star}, \sigma^{\star} = (Z^{\star}, Y^{\star}, Y^{\star'}, T^{\star}), \mathbf{D}^{\star}, nym_p^{\star}, \mathbf{W}^{\star})$ and conducting $ZKPoK(sk_P^{\star}, nym_P^{\star})$ then $\mathcal{B}$ extracts from the proof of knowledge contained in the Show protocol the value $sk_p^{\star}$ related to the $nym_P^{\star}$ and stores all elements. Moreover, no credential owned by corrupt users can be valid on this set of messages $\mathbf{D}^{\star}$ (as $\mathcal{A}$ can win the unforgeability game). This means that, for any credential on (sk_i) with all $i = \perp$, we have $(\mathbf{D}^{\star}, M') \notin \mathcal{R}_{k'}$. In all cases, this means that $(sk_P^{\star}, (\mathbf{C}^{\star}, D^{\star}, \mathbf{W}^{\star}), \sigma^{\star})$ is a valid forgery against our signature scheme, $\mathcal{B}$ breaks thus unforgeability of SPSEQ-VC which concludes our proof.

Step2. What we need to prove next is that the credentials that have revoked credentials in the delegation chain cannot pass verification. Adversary $\mathcal{A}$ manages to conduct a showing protocol accepted by the verifier using some revoked credentials or other revoked credentials in the delegation chain $\sigma_i \in A_X, i \in [L]$. Then, we construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the collision freeness of the accumulator scheme ACC.

Here, $\mathcal{B}$ is composed of an adversary $\mathcal{A}$ and a challenger $\mathcal{S}$ in the context of an unforgeability game. $\mathcal{B}$ interacts with a challenger $\mathcal{C}$ in a collision-resistant game

for the accumulator scheme *Acc*. Subsequently, we describe how $\mathcal{S}$ simulates the environment for $\mathcal{A}$ and interacts with challenger $\mathcal{C}$.

$\mathcal{B}$ interacts with $\mathcal{C}$ to obtain $PK_{ACC} = (BG, g_1^{s_i}, g_2^{s^i} | 0 \leq i \leq n)$. Then $\mathcal{S}$ runs $(pp, sk_{CA}, pk_{CA}) \leftarrow Setup(1^\lambda, 1^n, 1^l)$. $\mathcal{S}$ executes $\mathcal{A}(pp, pk_{CA}, PK_{ACC})$ and emulates all oracles as in the actual game. If $\mathcal{A}$ outputs $(cred_i, \pi, A_i)$, then $\mathcal{S}$ interacts with $\mathcal{A}$ as verifier. If $\mathcal{A}$ presents a valid display using credential $cred_i$, $\mathcal{S}$ rewinds $\mathcal{A}$ to the step after sending commitments in PoK and restarts $\mathcal{A}$ with a new challenge. This step allows the retrieval of the used non-membership witness π and the credential randomizer μ, by extracting the corresponding discrete logarithms. This ensures that the algorithm can backtrack to a known state and has a new challenge, preventing potential adversaries from exploiting the state of the system. $\mathcal{S}$ now computes $cred_i = cred'_i \cdot \mu^{-1}$ on the message part of the credential. If there is no such credential, $\mathcal{B}$ aborts. If $\forall j \in [1, i], cred_j \notin \mathcal{R}_{cred}$, then $\mathcal{S}$ aborts. Otherwise, it is known that the extracted witness π yields a correct evaluation of the verification relation, even though there is a revoked credential in the relevant credential chain. Therefore, $\mathcal{B}$ outputs π as a non-membership witness for an accumulated value, giving a collision for the accumulator.

(Anonymity). If the DDH assumption holds and the SPSEQ-VC provides *Origin-hiding, Conversion-privacy and Derivation-privacy*, then the DAAC-CR is anonymous.

Proof. We employ a sequence of games to demonstrate the anonymity of DAAC-CR, where each game is indistinguishable from the previous one. Henceforth, we denote by S_i the event that the opponent wins the *Game* i.

Game 0: The original game as given in Definition x.

Game 1: As *Game* 0, in addition to $\mathcal{A}$ outputting pk_{CA} and the corresponding NIZK(sk_{CA}, pk_{CA}), the experimental execution of the NIZK knowledge extractor is performed. It extracts a witness $((x_i)_{i\in[0,l]}, (x'_i)_{i\in[1,l]})$ as well as the trapdoors α, s, which are set as sk_{CA}. If the extractor fails, we abort.

Transition 1 - Game 1 → Game 2: The success probability in Game 1 is the same as in Game 0, unless the knowledge extractor fails, we have that $\Pr[S_1] - \Pr[S_0] \leq \epsilon(\lambda)$.

Game 2: As *Game* 1, except that the experiment runs $\mathcal{O}^{Ano-b}$ as follows: Like in *Game* 1, but all executions of $Random_{All}$ for the credential $(i, uk_{k'}, \mathbf{A}, \sigma) \leftarrow \mathcal{L}_{cred}[j_b]$ are replaced by *Sign*. Oracles are simulated as in *Game* 1, except for the following oracles as: $\mathcal{O}^{RIO}$: all executions of *Delegate* and *ConvertSig* for credentials $(i, uk_{k'}, \mathbf{A}, \sigma) \in \mathcal{L}_{cred}$ are replaced by *Sign*.

Transition 1 - Game 1 → Game 2: By Origin-hiding, Derivation-privacy and Conversion-privacy, replacing signatures from $Random_{All}$, *Delegate* and *ConvertSig* with ones from *Sign* are identically distributed for all (**AC**). We have that $\Pr[S_1] = \Pr[S_2]$.

Game 3: As *Game* 2, except that the experiment runs $\mathcal{O}^{Ano-b}$ as follows: All proofs $ZKPok(sk_P, nym_P)$ in *Show*, are simulated.

Transition 3 - Game 2 → 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: As *Game* 3, except for the following changes. Let q_u be the number of queries made to $\mathcal{O}^{HU}$. At the beginning *Game* 4 picks $\omega \leftarrow [q_u]$ and runs $\mathcal{O}^{HU}$,$\mathcal{O}^{CU}$ and $\mathcal{O}^{Ano-b}$ as follows:

$\mathcal{O}^{HU}(i)$: As in *Game* 3, except if this is the ω-th call to the oracle then it additionally defines $i^\star \leftarrow i$.

$\mathcal{O}^{CU}(i, pk_u)$: If $i \in CU$ or $i \in \mathcal{O}^{Anon-b}$, it returns $\perp$. If $i = i^\star$ then the experiment stops and outputs a random bit $b' \leftarrow \{0, 1\}$. Otherwise, if $i \in HU$, it returns user $i's$ sk_i and credentials and moves i from HU to CU; and if $i \notin HU \cup CU$, it registers and adds i to CU a new corrupt user with public key pk_i.

$\mathcal{O}^{Ano-b}(j_0, j_1, D)$: As in *Game* 3, except if $i^\star \neq i_b \leftarrow \mathcal{L}_{cred}[j_b]$, the experiment stops and outputs $b' \leftarrow \{0, 1\}$

Transition 4 - Game 3 → Game 4: By assumption, $\mathcal{O}^{Ano-b}$ is called at least once with some input (j_0, j_1, D) such that $i_0 \leftarrow \mathcal{L}_{cred}[j_0], i_1 \leftarrow \mathcal{L}_{cred}[j_1] \in HU$. If $i^\star = i_b$ then $\mathcal{O}^{Ano-b}$ does not abort and neither does $\mathcal{O}^{CU}$. Since $i^\star = [i_b]$ with probability $\frac{1}{q_u}$, the probability thaft the experiment does not abort is at least $\frac{1}{q_u}$, and thus $\Pr[S_4] \geq (1 - \frac{1}{q_u})\frac{1}{2} + \frac{1}{q_u}\Pr[S_3]$.

Game 5: As *Game* 4, except $\mathcal{O}^{Ano-b}$: it picks $\mathbf{C} \leftarrow (\mathbb{G}_1^\star)^k$ and performs the showing using $cred' \leftarrow (\mathbf{C}, Sign)$, with corresponding $D = (d_i)_{i\in[k]}$ and W_i for $i \in [k]$. Note that the only difference is the choice of $\mathbf{C}$; while $\mathbf{W}$ is distributed as in *Game* 4, in particular, they are unique elements satisfying $VC.Verify(C_i, D_i, W_i), i \in [k]$. Finally, picks $A_X \leftarrow \mathbb{G}_1^\star, A_{ID} \leftarrow \mathbb{G}_1^\star, \bar{\pi} \leftarrow \mathbb{G}_1^\star \times \mathbb{G}_2^\star$, which satisfies $NonMemVerify_{ACC}$.

Transition 5 - Game 4 → Game 5: Let (BG, P_x, P_y, P_z) be a DDH instance for $BG = BG_{Gen}(1^\lambda)$ 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 $(P, P^{x_1}, ..., P^{x_k}, P^y, Z_1, ..., Z_k)$ where $Z_i = P^{x_i\cdot y}$ or random for all $\{1, ..., k\}$. One can easily show that this extended version of DDH follows from DDH itself as long as k is a polynomial. Oracles are simulated as in *Game* 4, except for the following oracles as:

$\mathcal{O}^{OIS}(i, \mathbf{A})$: As in *Game* 4, except for the computation of the following values if $i = i^\star$. Let this be the i-th call to this oracle. Since $\alpha \notin A_i$, it computes C_i for $A_i \in \mathbf{A}$.

$\mathcal{O}^{CSH}(i, D)$: As in *Game* 4, with the difference that if$i^\star = i \leftarrow \mathcal{L}_{cred}[j]$, it computes the witness W_i and $\bar{\pi}$.

$\mathcal{O}^{Ano-b}(j_0, j_1, D)$: As in *Game* 4, with the following difference. Using self-reducibility of DDH, it picks $s, t \leftarrow \mathbb{Z}_p^\star$ and computes $Y' \leftarrow P^{t\cdot y} \cdot P^s = P^{y'}$ with $y' \leftarrow t \cdot y + s$, and $Z_i' \leftarrow P^{t\cdot z_i} \cdot P^{s\cdot x_i} = P^{(t(z_i - x_i\cdot y) + x_i\cdot y')}$. It performs the showing using the following values. Since $a \notin D : C_i \leftarrow VC.commit(A_i) \cdot Z_i'$, $W_i \leftarrow VC.Prove$ and $\bar{\pi} \leftarrow NonMemProve_{ACC}$.

Apart from an error event happening with negligible probability, we have simulated *Game* 4 if the DDH instance was "real" and *Game* 5 otherwise. If during the simulation of $\mathcal{O}^{Ano-b}$ it occurs that any $Y_i' = 0_{\mathbb{G}_1}$ or $Z_i' = 0_{\mathbb{G}_1}$ then the distribution of values is not as in one of the two games. In case of a DDH instance, we have for all $C_i \leftarrow VC.commit(A_i) \cdot y'$; otherwise all C_i are

independently randoms. Let $\epsilon_{DDH}(\lambda)$ denote the advantage of solving the DDH problem and q_l the number of queries to the $\mathcal{O}^{Ano-b}$, we have $|\Pr[S_4]-\Pr[S_5]| \leq \epsilon_{DDH}(\lambda) + (1+2q_l)\frac{1}{p}$.

In *Game* 5 the $\mathcal{O}^{Ano-b}$ oracle returns a fresh signature σ on random elements $\mathbf{C} \leftarrow (\mathbb{G}_1^\star)^k$ 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

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

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

$$\Pr[S_0] \leq \Pr[S_1] + k\cdot s(\lambda)\cdot\frac{1}{2} + k\cdot s(\lambda) + q_u\cdot(\epsilon_{DDH}(\lambda) + (1+2q_u)\cdot\frac{1}{p}).$$

where $\Pr[S_1] = \Pr[S_3]$; q_u, q_o and q_l are the number of queries to the $\mathcal{O}^{HU}, \mathcal{O}^{COB}$ and the $\mathcal{O}^{Ano-b}$ oracle, respectively. Assuming security of $ZKPoK$, $NIZK$ and DDH, the adversary's advantage is thus negligible.

References

1. Camenisch J, Dubovitskaya M, Lehmann A, et al. Concepts and languages for privacy-preserving attribute-based authentication[C]//Policies and Research in Identity Management: Third IFIP WG 11.6 Working Conference, IDMAN 2013, London, UK, April 8-9, 2013. Proceedings 3. Springer Berlin Heidelberg, 2013: 34-52.
2. Camenisch J, Lysyanskaya A. Signature schemes and anonymous credentials from bilinear maps[C]//Annual international cryptology conference. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004: 56-72.
3. Chaum D. Blind signatures for untraceable payments[J]. 1983.
4. Hanser C, Slamanig D. Structure-preserving signatures on equivalence classes and their application to anonymous credentials[C]//Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20. Springer Berlin Heidelberg, 2014: 491-511.
5. Camenisch J, Lysyanskaya A. An efficient system for non-transferable anonymous credentials with optional anonymity revocation[C]//Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20. Springer Berlin Heidelberg, 2001: 93-118.
6. Connolly A, Lafourcade P, Perez Kempner O. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes[C]//IACR International Conference on Public-Key Cryptography. Cham: Springer International Publishing, 2022: 409-438.

7. Chase M, Lysyanskaya A. On signatures of knowledge[C]//Advances in Cryptology-CRYPTO 2006: 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006. Proceedings 26. Springer Berlin Heidelberg, 2006: 78-96.
8. Belenkiy M, Camenisch J, Chase M, et al. Randomizable proofs and delegatable anonymous credentials[C]//Advances in Cryptology-CRYPTO 2009: 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings. Springer Berlin Heidelberg, 2009: 108-125.
9. Camenisch J, Drijvers M, Dubovitskaya M. Practical UC-secure delegatable credentials with attributes and their application to blockchain[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 683-699.
10. Crites E C, Lysyanskaya A. Delegatable anonymous credentials from mercurial signatures[C]//Cryptographers' Track at the RSA Conference. Cham: Springer International Publishing, 2019: 535-555.
11. Mir O, Slamanig D, Bauer B, et al. Practical delegatable anonymous credentials from equivalence class signatures[J]. Proceedings on Privacy Enhancing Technologies, 2023.
12. Berrut J P, Trefethen L N. Barycentric lagrange interpolation[J]. SIAM review, 2004, 46(3): 501-517.
13. von zur Gathen J, Gerhard J. Fast polynomial evaluation and interpolation[J]. Modern Computer Algebra, 2013: 295-310.
14. Srinivasan S, Karantaidou I, Baldimtsi F, et al. Batching, Aggregation, and Zero-Knowledge Proofs in Bilinear Accumulators[C]//Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. 2022: 2719-2733.
15. Kate A, Zaverucha G M, Goldberg I. Constant-size commitments to polynomials and their applications[C]//Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16. Springer Berlin Heidelberg, 2010: 177-194.
16. Tomescu A, Abraham I, Buterin V, et al. Aggregatable subvector commitments for stateless cryptocurrencies[C]//International Conference on Security and Cryptography for Networks. Cham: Springer International Publishing, 2020: 45-64.
17. Abe M, Fuchsbauer G, Groth J, et al. Structure-preserving signatures and commitments to group elements[C]//Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30. Springer Berlin Heidelberg, 2010: 209-236.
18. Crites E C, Lysyanskaya A. Mercurial Signatures for Variable-Length Messages[J]. Proceedings on Privacy Enhancing Technologies, 2021, 4: 441-463.
19. Blömer J, Bobolz J. Delegatable attribute-based anonymous credentials from dynamically malleable signatures[C]//International Conference on Applied Cryptography and Network Security. Cham: Springer International Publishing, 2018: 221-239.
20. Chase M, Kohlweiss M, Lysyanskaya A, et al. Malleable signatures: New definitions and delegatable anonymous credentials[C]//2014 IEEE 27th computer security foundations symposium. IEEE, 2014: 199-213.
21. Groth J, Sahai A. Efficient non-interactive proof systems for bilinear groups[C]//Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27. Springer Berlin Heidelberg, 2008: 415-432.

22. Hanzlik L, Slamanig D. With a little help from my friends: Constructing practical anonymous credentials[C]//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. 2021: 2004-2023.
23. Bogatov D, De Caro A, Elkhiyaoui K, et al. Anonymous transactions with revocation and auditing in hyperledger fabric[C]//Cryptology and Network Security: 20th International Conference, CANS 2021, Vienna, Austria, December 13-15, 2021, Proceedings 20. Springer International Publishing, 2021: 435-459.
24. Crites E, Kohlweiss M, Preneel B, et al. Threshold structure-preserving signatures[C]//International Conference on the Theory and Application of Cryptology and Information Security. Singapore: Springer Nature Singapore, 2023: 348-382.
25. Damgård I. Efficient concurrent zero-knowledge in the auxiliary string model[C]//International Conference on the Theory and Applications of Cryptographic Techniques. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000: 418-430.
26. Shoup V. Lower bounds for discrete logarithms and related problems[C]//Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16. Springer Berlin Heidelberg, 1997: 256-266.
27. Bogatov D, De Caro A, Elkhiyaoui K, et al. Anonymous transactions with revocation and auditing in hyperledger fabric[C]//Cryptology and Network Security: 20th International Conference, CANS 2021, Vienna, Austria, December 13-15, 2021, Proceedings 20. Springer International Publishing, 2021: 435-459.
28. Begum N, Nakanishi T. An accumulator-based revocation in delegatable anonymous credentials[C]//2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW). IEEE, 2020: 314-320.
29. Li P, Lai J, Yang Y, et al. Attribute-based anonymous credential: Delegation, traceability, and revocation[J]. Computer Networks, 2023, 237: 110086.
30. Fuchsbauer G, Hanser C, Slamanig D. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials[J]. Journal of Cryptology, 2019, 32: 498-546.